Policy-based planning for student mobility support in e-Learning systems

PhD Thesis

Pavel Nikolaev

Software Technology Research Laboratory
Faculty of Technology
De Montfort University
England

This thesis is submitted in partial fulfillment of the requirements for the Doctor of Philosophy.

2014

Declaration

I declare that the work described in this thesis is original work undertaken by me during my period of registration for the degree of Doctor of Philosophy at the Software Technology Research Laboratory (STRL), Faculty of Technology, De Montfort University, United Kingdom. It is submitted for the degree of Doctor of Philosophy at De Montfort University.

Acknowledgement

First of all, I would like to thank people who helped me to start the PhD programme at De Montfort University. I would like to heartily thank Prof. Alexander Chernikov and Prof. Jeff Knight for this opportunity. The PhD studies had great impact on my personal and professional development and changed my vision of life and my place in it irreversibly.

I would like to express my sincere gratitude to all members of my supervisory team, who worked with me during these years, contributed to this work and to my development. I am deeply grateful to my first supervisor, Dr. Aladdin Ayesh, for his encouragement, his wisdom guidance and advises, which helped me to advance in my work. Again, I would like to sincerely thank Prof. Alexander Chernikov, as my second supervisor, for all his ideas, support and help, which I received from him during all time of the research. Last but not the least, I would like to express my deep appreciation to Prof. Hussein Zedan, the head of STRL, for his valuable contribution to this research, his huge experience and wisdom, which he shared with me, and for his the utmost encouragement and inspiration.

I would like to deeply thank all members of my family, who supported and encouraged me during my studies. During this research there were a lot of challenges that required highly concentrated efforts. Without the comprehensive support and concern of my closest people, I would not be able to cope with them and finish what was required. I thank my parents, Evgeny Nikolaev and Svetlana Nikolaeva, my girlfriend, Yelena Chernikova, and my grandmother, Irina Nikolaeva. I also would like to pass many thanks to all colleagues, which I worked with within STRL and outside it, and all people, who made this project possible and assisted me with it.

Publications

1. Pavel Nikolaev, Aladdin Ayesh and Elena Chernikova. Policy-based HTN planning for combined course programmes generation. In *Proceedings of the Fourth International Conference on Internet Technologies and Applications (ITA 11)*, pages 457 - 464. Glyndwr University, UK, 2011.
2. Pavel Nikolaev and Aladdin Ayesh. Combined course programmes generation in multi-agent e-Learning system using policy-based HTN planning. In James O'Shea, Ngoc Nguyen, Keeley Crockett, Robert Howlett and Lakhmi Jain, editors, *Agent and Multi-Agent Systems: Technologies and Applications*, vol. 6682 of *Lecture Notes in Coputer Science*, pages 504-513. Springer Berlin / Heidelberg, 2011.
3. Pavel Nikolaev and Alexander Chernikov. Policy-based planning in a multi-domain hierarchical environment. *Herald of Bauman Moscow State Technical University*, Special Issue *Computer Engineering*, pages 76 - 80. BMSTU Publishing House, Moscow, Russia, 2011.
4. Pavel Nikolaev, Aladdin Ayesh and Hussein Zedan. Combined Course Programmes Support System (CProgS). In *Proceedings of the Second Creativity, Innovation and Software Engineering conference (CISE'09)*, Ravda, Bulgaria, 2009.
5. Pavel Nikolaev and Aladdin Ayesh. Development of a System for Combined Course Programmes Support. In *Book of abstracts presented on Russian Scientific-Technicall Conference "Manufactoring Enginering Technologies" with international participation*, pages 283 - 284. BMSTU Publishing House, Moscow, Russia, 2008.

Abstract

Student mobility in the area of Higher Education (HE) is gaining more attention nowadays. It is one of the cornerstones of the Bologna Process being promoted at both national and international levels. However, currently there is no technical system that would support student mobility processes and assist users in authoring educational curricula involving student mobility. In this study, the problem of student mobility programmes generation based on existing modules and programmes is considered. A similar problem is being solved in an Intelligent Tutoring Systems field using Curriculum generation techniques, but the student mobility area has a set of characteristics limiting their application to the considered problem. One of main limiting factors is that mobility programmes should be developed in an environment with heterogeneous regulations. In this environment, various established routines and regulations are used to control different aspects of the educational process. These regulations can be different in different domains and are supported by different authors independently.

In this thesis, a novel framework was developed for generation of student mobility programmes in an environment with heterogeneous regulations. Two core technologies that were coherently combined in the framework are hierarchical planning and policy-based management. The policy-based planner was designed as a central engine for the framework. It extends the functionality of existing planning technologies and provides the means to carry out planning in environments with heterogeneous regulations, specified as policies. The policy-based planner enforces the policies during the planning and guarantees that the resultant plan is conformant with all policies applicable to it. The policies can be supported by different authors independently. Using them, policy authors can specify additional constraints on the execution of planning actions and extend the pre-specified task networks. Policies are enforced during the planning in a coordinated manner: situations when a policy can be enforced are defined by its scope, and the outcomes of policy evaluation are processed according to the specially defined procedures.

For solving the problem of student mobility programme generation using the policy-based planner, the planning environment describing the student mobility problem area was designed and this problem was formalised as a planning task. Educational processes valid throughout the HE envi-

ronment were formalised using Hierarchical Task Network planning constructs. Different mobility schemas were encoded as decomposition methods that can be combined to construct complex mobility scenarios satisfying the user requirements. New mobility programmes are developed as detailed educational processes carried out when students study according to these programmes. This provides the means to model their execution in the planning environment and guarantee that all relevant requirements are checked.

The postponed policy enforcement mechanism was developed as an extension of the policy-based planner in order to improve the planning performance. In this mechanism, future dead-ends can be detected earlier during the planning using partial policy requests. The partial policy requests and an algorithm for their evaluation were introduced to examine policies for planning actions that should be executed in the future course of planning. The postponed policy enforcement mechanism was applied to the mobility programme generation problem within the descending policy evaluation technique. This technique was designed to optimise the process of programme components selection. Using it, policies for different domains can be evaluated independently in a descending order, gradually limiting the scope for the required component selection.

The prototype of student mobility programme generation solution was developed. Two case studies were used to examine the process of student mobility programmes development and to analyse the role of policies in this process. Additionally, four series of experiments were carried out to analyse performance gains of the descending policy evaluation technique in planning environments with different characteristics.

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Problem statement 3

1.3 Research questions 5

1.4 Research methodology 6

1.5 Scope of the research 8

1.6 Success criteria 9

1.7 Original contributions 10

1.8 Thesis organisation 12

2 Literature review **14**

2.1 Introduction 14

2.2 e-Learning systems 15

2.2.1 Virtual Learning Environments (VLEs) 15

2.2.2 Intelligent tutoring systems (ITSs) 17

2.2.2.1 ITS architecture 17

2.2.2.2 ITS main techniques 20

2.2.2.3 Curriculum generation techniques 20

2.2.3 Planning technologies for Curriculum generation 26

2.2.3.1 Hierarchical Task Network (HTN) planning 27

2.2.3.2 Planning technologies for Curriculum generation and Combined Educational Programmes development 29

2.2.4 Multi-agent e-Learning systems (MASs) 34

2.3 Policy-based management 35

2.3.1 Policy definition 35

2.3.2 Benefits of the policy-based management 36

2.3.3 Types of policy rules 37

2.3.3.1 Authorisation rules 38

2.3.3.2 Reactive rules 38

2.3.3.3 Role-assignment rules 39

2.3.3.4 Routing rules 39

2.3.3.5 Delegation rules 39
2.3.3.6 Obligation rules support 40
2.3.3.7 Rule types analysis 41
2.3.4 Policy enforcement 42
2.3.5 Policy composition 44
2.3.6 Policy languages 45
2.3.6.1 Ponder 45
2.3.6.2 KAoS 46
2.3.6.3 EPAL 46
2.3.6.4 XACML 47
2.3.6.5 Policy languages comparison 48
2.4 Conclusion 48

3 Framework for student mobility programmes development 51
3.1 Introduction 51
3.2 Student mobility domain area analysis 52
3.2.1 Student mobility processes analysis 52
3.2.2 International initiatives in the area of higher education 54
3.2.3 Other characteristics of student mobility domain area 56
3.3 Requirements to Combined Educational Programme development solution 57
3.4 Combined educational programmes generation framework 59
3.4.1 Specification layer 60
3.4.2 Processing layer 62
3.4.3 Data layer 62
3.4.4 An overview of the CEP generation process 63
3.5 Conclusion 65

4 XACML policy specification language formalisation 66
4.1 Introduction 66
4.2 XACML policy language overview 68
4.3 Policy evaluation schema 70
4.4 Abstract syntax of XACML policies 72
4.5 A formal model of policy evaluation 73
4.5.1 Policy set evaluation 74
4.5.2 Policy and policy set combining algebras 75
4.5.3 Policy and rule evaluation 78
4.5.4 Rule combining algebras 79

4.5.5 Target and condition evaluation . . . 80
4.5.5.1 Target evaluation . . . 82
4.5.5.2 Condition evaluation . . . 83
4.6 Obligations generation during policy evaluation . . . 84
4.7 Conclusion . . . 85

5 Policy-based planner **86**
5.1 Introduction . . . 86
5.2 Policy-based planner overview and general processes . . . 88
5.2.1 Conceptual model . . . 89
5.2.2 Main interaction processes . . . 91
5.2.2.1 Actions legitimacy and policy evaluation . . . 91
5.2.2.2 Extensions of planning domain using policy obligations . . . 93
5.2.2.3 Conditional plans construction using policy conditions . . . 94
5.3 Planning . . . 95
5.3.1 The planner's world state and its object model . . . 95
5.3.2 Planning domain specification . . . 97
5.3.2.1 Tasks . . . 97
5.3.2.2 Operators . . . 98
5.3.2.3 Methods . . . 101
5.3.3 Plan representations . . . 103
5.3.4 Obligations processing . . . 105
5.3.5 Planning algorithm . . . 106
5.4 Policies . . . 108
5.4.1 Policy request . . . 108
5.4.2 Policy specification . . . 112
5.4.2.1 Conditions specification . . . 113
5.4.2.2 Time constraints . . . 113
5.4.2.3 Obligations specification . . . 114
5.4.3 Obligations validation mechanism . . . 115
5.5 Transformation rules engine . . . 118
5.6 Adaptive object contexts generation technique for policy request construction . . . 119
5.6.1 Hyper-graph of planner's world state and its object model . . . 120
5.6.2 Abstract contexts . . . 123
5.6.3 Generation of object contexts . . . 125
5.7 Conclusion . . . 128

6 Postponed policy enforcement mechanism **130**

6.1 Introduction 130

6.2 Postponed policy enforcement 132

6.2.1 Constructs for partially known information specification 132

6.2.2 Partial policy requests 134

6.2.2.1 Constructs for future planner's states projection 135

6.2.2.2 Partial policy requests generation 138

6.2.3 Operators and methods execution routines 139

6.2.4 Postponed policy enforcement 142

6.3 Partial policy evaluation 145

6.3.1 Requirements to partial policy evaluation 145

6.3.2 Policy set evaluation function 148

6.3.3 Policy and policy set combining algebras 149

6.3.4 Policy and rule evaluation functions 151

6.3.5 Rule combining algebras 153

6.3.6 Target and condition evaluation functions 155

6.3.6.1 Information retrieval from partial policy request tuples 155

6.3.6.2 Target evaluation and truth-value functions with Indeterminate Temporal value support 158

6.3.6.3 Condition evaluation functions 160

6.3.7 Loose time intervals processing 165

6.4 Conclusion 165

7 Planning for CEP development **167**

7.1 Introduction 167

7.2 Learning objects and their relations specification 168

7.2.1 Learning objects specification 168

7.2.2 Learning outcomes-based relations between Learning objects 172

7.2.3 Hierarchical multi-domain structure and policies for Learning objects 173

7.2.4 Transformation rules for Learning objects properties 176

7.3 Planning procedures for CEP development 177

7.3.1 HTN planning domain for CEP generation process 177

7.3.1.1 Input requirements for CEP generation 177

7.3.1.2 BTr development phase 179

7.3.1.3 BTr validation phase 185

7.3.1.4 Low-level routines 187

7.3.1.5 Variations of overall CEP construction procedure 191
7.3.2 Descending policy evaluation technique 192
7.3.2.1 Utilisation of postponed policy enforcement for CEP generation problem 192
7.3.2.2 Descending policy evaluation algorithm 194
7.3.2.3 Domain refinement 196
7.3.2.4 Algorithms for planning with domain refinement 200
7.4 Conclusion 204

8 Implementation **205**
8.1 Introduction 205
8.2 Planner module 209
8.2.1 Custom call-terms for policy-based planner implementation 210
8.2.2 Planner's world state object model 211
8.3 Policy evaluation mediator 212
8.4 Policy analyser 215
8.5 Policy evaluator 220
8.6 Transformation rules evaluator 223
8.7 Conclusion 223

9 Evaluation **224**
9.1 Introduction 224
9.2 Case studies 224
9.2.1 Case study 1 224
9.2.2 Case study 2 239
9.3 Planning in environments with heterogeneous regulations 249
9.3.1 Enforcement of relevant policy constraints 250
9.3.2 Enforcement of established routines 253
9.3.3 Planning in an environment with dynamically changing regulations 255
9.4 Performance analysis 256
9.4.1 Policies characteristics impact analysis 257
9.4.2 Domain tree characteristics impact analysis 263
9.5 Conclusion 269

10 Conclusion **271**
10.1 Summary 271
10.2 Original contributions 272

10.3 Revisiting success criteria 275
10.4 Future work 276

Appendix A Evaluation of monotonicity for Partial policy evaluation algorithm 295

Appendix B CEP generation planning domain specification 302

Appendix C ANTLR grammar for XPath parsing and AST generation 312

Appendix D Case study 2 planning environment specifications 315

Appendix E Experimental results for performance analysis 324

List of Tables

2.1 Policy languages comparison . . . 50

4.1 Table of values for Policy evaluation function P^e . . . 75
4.2 Tables of values for Permit-overrides and Deny-overrides policy combining operations $\bullet_p^{PO}$ and $\bullet_p^{DO}$ . . . 76
4.3 Table of values for Rule evaluation function R^e . . . 79
4.4 Tables of values for permit- and deny-overrides rule combining operations $\bullet_r^{PO}$ and $\bullet_r^{DO}$ . . . 80
4.5 Type of XACML functions according to their abstract signatures . . . 84

5.1 Interpretation of policy decisions by the planning engine . . . 92

6.1 Table of values for Policy evaluation function P^{ep} . . . 148
6.2 Tables of values for permit- and deny-overrides policy combining operations $\bullet_{pp}^{PO}$ and $\bullet_{pp}^{DO}$ . . . 150
6.3 Table of values for rule evaluation function R^{ep} . . . 152
6.4 Table of values for permit-overrides and deny-overrides rule combining operations $\bullet_{rp}^{PO}$ and $\bullet_{rp}^{DO}$ . . . 153
6.5 "$A \wedge^p B$" and "$A \vee^p B$" operations definitions in $TRVal^p$ . . . 159
6.6 "$\neg^p A$" operation definition in $TRVal^p$ . . . 159
6.7 Tables of values for True- and False-preserving correction functions γ_{Tr} and γ_{Fl} and Constant correction function γ_{All} . . . 164

9.1 Specification of constants, used in policy rules for different policies (case study 1) . 228
9.2 Minimum branching factors for tasks estimated before planning step execution . . 237
9.3 Three versions of the planning algorithm comparison (case study 1) . . . 238
9.4 CEP structure generated (case study 2) . . . 248
9.5 Policy decisions received during execution of different scenarios . . . 251
9.6 Policy decisions received during execution of different scenarios (cont.) . . . 252

List of Figures

2.1 Generalised models and functions of ITS 18
2.2 Conceptual graph, student and learning objects models of ABITS 21
2.3 Types of policy rules 38
2.4 Obligation types 41
2.5 Outsourced (A.) [6], p. 54 and provisional (B.) [103], p. 6 policy enforcement models 43

3.1 Student mobility schemas 53
3.2 CEP generation framework 59
3.3 Overview of the CEP generation process 64

4.1 An overall schema of policy evaluation 71
4.2 Abstract syntax for XACML policies 72
4.3 Example of AST 73
4.4 Abstract syntax for XACML policies (cont.) 81
4.5 All possible transformations between abstract data types 84

5.1 Overview schema of the policy-based planner 88
5.2 Conceptual model for action 89
5.3 Example of hierarchical plan generation 104
5.4 Different variants of task execution 106
5.5 Planning algorithm 109
5.6 Planning algorithm (cont.) 110
5.7 Example policy structure 112
5.8 Examples of policies, applicable to an action 114
5.9 Syntax for validation rules 116
5.10 Examples of validation rules 118
5.11 Example of the planner's world state 121
5.12 Hyper-graph of the planner's world state example 121
5.13 Object model hyper-graph for the example planner's world state 122
5.14 Policy request generation schema using abstract contexts 125
5.15 Object context generation algorithm 126
5.16 Transformation of Bipartite graph for $k(\mathbf{H}_{Obj})$ into an object context 128

6.1 Example of compound task decomposition method $meth_{CT_i}$ execution 137
6.2 Correctness checks for compound action decomposition method execution 141
6.3 Partial policy requests evaluation algorithm 144
6.4 Partial policy evaluation process 146
6.5 Approximation order on set M_1^p 146
6.6 Approximation order on $TRVal^p \times M_1^p$ set 149
6.7 Approximation order on set M_2^p (A.) Graph for decisions mapping function f_p : $M_2^p \rightarrow M_1^p$ (B.) 151
6.8 Correction mechanism, applied to function '*Func*' 163

7.1 Hierarchical multi-domain environment (domain tree) 174
7.2 Examples of different decomposition methods application 185
7.3 Obligation validation rules 189
7.4 Basic descending policy evaluation algorithm 195
7.5 State transition diagram for task states during the planning 199
7.6 Ordered planning algorithm with domain refinements 201
7.7 Unordered version of the planning algorithm with domain refinements 203

8.1 Prototype architecture 208
8.2 Class diagram for the planner's world state 212
8.3 Example of the planner's world state 213
8.4 Class diagram for the policy evaluation mediator 215
8.5 Parsing/rewrite rules for processing location paths and generation of AST 217
8.6 Examples of parsing/rewrite rules that introduce AST branching 218
8.7 XPath expression, corresponding AST and abstract context tree 219
8.8 Algorithm for Abstract context generation from AST 221

9.1 Domain tree schema (case study 1) 225
9.2 Schemas for $Country_1$ and Uni_{11} policies (case study 1) 227
9.3 Initial planner's world state (case study 1) 229
9.4 Abstract contexts for university and country policies (case study 1) 229
9.5 Policy request for &*study_interval* action. Original policy-based planning. 231
9.6 Task network decompositions structure. Original policy-based planning. 232
9.7 Task network decompositions structure. Descending policy evaluation. 232
9.8 Partial policy request for &*study_interval* action in $Country_1$ domain. Descending policy evaluation. 235
9.9 Domain tree schema (case study 2) 240

9.10 High-level policy schema (case study 2) 242
9.11 Schema for *Russia* policy set (case study 2) 243
9.12 Task network decompositions structure in case study 2 245
9.13 Task network decompositions structure in case study 2 (cont.) 246
9.14 CEP process model generated (case study 2) 249
9.15 Task network representing one semester study period, generated within the BMSTU domain 254
9.16 Task network representing one semester study period, generated within the University of Birmingham domain 255
9.17 Experimental results for different values of policy stringency p 259
9.18 CPU time ratios for different values of policy stringency p 260
9.19 Experimental results for different values of z (see also Figure E.2) 262
9.20 Experimental results for different number of EPs (see also Figure E.3) 264
9.21 CPU time ratios for different number of EPs 265
9.22 Experimental results for different values of K_{br} (task with 2 levels) 266
9.23 Experimental results for number of levels N_{Lev} ($K_{br} = 2$) 267

A.1 Approximation order for $TRVal^p \times TRVal^p$ set 295

D.1 Schema for *Russia* policy set 320
D.2 Schemas for $BMSTU$ and EP_3 policy sets 321
D.3 Schemas for UK and *University of Birmingham* policy sets 322
D.4 Schemas for *School of Computer Science, University of Birmingham* and Msc in Advanced Computer Science EP_1 policy sets 323

E.1 Examples of domains trees used for performance analysis 324
E.2 Experimental results for different values of z (Experiment 1.2) 325
E.3 Experimental results for different number of EPs (Experiment 2.1) 326

Acronyms

ACM	Association for Computing Machinery
AI	Artificial Intelligence
AST	Abstract Syntax Tree
BMSTU	Bauman Moscow State Technical University
BP	Bologna Process
BSc	Bachelor of Science
BTr	Basic Track
CEFR	Common European Framework of Reference for Languages
CEP	Combined Educational Programme
CG	Curriculum Generation
CPU	Central Processing Unit
DAAD	German Academic Exchange Service (Deutscher Akademischer Austauschdienst)
DE	Decreasing Effects
DO	Deny-Overrides
ECA	Event-Condition-Action
ECTS	European Credit Transfer and Accumulation System
EHEA	European Higher Education Area
EHEA QF	Qualifications Framework of the European Higher Education Area
EP	Educational Programme
FAF	Fewest-Alternatives First
GUI	Graphical User Interface
HE	Higher Education
HTN	Hierarchical Task Network
IE	Increasing Effects
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IEEE LOM	Institute of Electrical and Electronics Engineers Learning Object Metadata
IETF	Internet Engineering Task Force
ISCED	International Standard Classification of Education
ISO	International Organisation for Standardisation
IT	Information Technologies
ITr	Initial Track
ITS	Intelligent Tutoring System

KhPU	Kharkov Polytechnic University
LCA	Least Common Ancestor
LObj	Learning Object
LOR	Learning Objects Repository
MAS	Multi-Agent System
MSc	Master of Science
NQF	National Qualification Framework
OASIS	Organisation for the Advancement of Structured Information Standards
PC	Policy Consumer
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PO	Permit-Overrides
PT	Policy Target
TAL	Temporal Action Logic
TO	Teaching Operator
UNESCO	United Nations Educational, Scientific and Cultural Organization
VLE	Virtual Learning Environment
W3C	World Wide Web Consortium
XACML	eXtensible Access Control Markup Language
XML	Extensible Markup Language
XPath	XML Path Language

Chapter 1

Introduction

1.1 Motivation

Student mobility in the area of Higher Education (HE) is the possibility of students to change their education providers (or courses) one or more times within their programmes of study in HE. Student mobility is one of the cornerstones of the Bologna Process (BP) and is promoted at national and international levels [58, 165]. It is beneficial for students and universities, as the mobility provides students with better possibilities for professional and personal development, and makes them more socially adaptable and self-confident. It increases the employability of students in their home countries as well as overseas and satisfies the demands of the global market [143, 165].

The UK Higher Education Statistics Agency defines two major types of student mobility [91]: **diploma mobility** is a mobility for an entire programme of study and **credit mobility** is a mobility for a part of the programme. The third distinguished type of mobility, viz., **voluntary moves**, covers other moves undertaken for a range of personal reasons. This type of student mobility is different from the previous types as it usually does not involve a recognition of study periods. The recognition of study periods during the student mobility is obviously profitable and desirable for students and society [86], as it reduces the overall expenses for achieving the students' educational goals and makes mobile students' experience smoother. Accordingly, the focus of this study will be on the first and second types of student mobility. The recognition of study periods will be considered as an integral part of the student mobility programme design.

Another classification of student mobility divides all the mobility schemas into internal, external and international mobility. **Internal student mobility** occurs when a student changes his (or her) course within the same education provider (e.g., a university). **External student mobility** occurs when a student changes the education provider within the same country. **International student mobility**, as UNESCO defines, "*implies a period of study in a country other than a student's country of residence ('the home country')*"[1].

However, a number of obstacles were identified [86, 57, 63, 165] preventing an intensive development of the student mobility area and a rapid growth of mobile student numbers:

Information. For the development of mobility programmes, detailed information about different

[1] www.unesco.org/education/studyingabroad/what_is/mobility.shtml [Accessed 23.02.2014].

aspects of the HE systems in other countries is needed, including regulations at the national and university levels, and student mobility initiatives. Students require information about available educational courses and student mobility programmes. Universities require information about their partner education providers. Available information is broad, heterogeneous and unstructured. Its analysis, for example, for the establishment of correspondence between the educational systems or specific modules, requires substantial efforts and deep expertise.

Financial. The development of student mobility programmes leads to an increase of expenses. Students can suffer from extra expenses for education in a university abroad. Universities can lose tuition fees from students who transfer to other universities. Finally, the design of student mobility programmes requires substantial financial investments from the universities. Moreover, as rigid regulations can exist in the student mobility area and the programmes designed should conform to them, the task of mobility programme design becomes even more complex and expensive.

Academic. The recognition of academic qualifications and credits, carried out during or after the student mobility activities, is a highly labour-intensive process. It requires substantial time and deep knowledge of the educational regulations and the problem area. During the recognition, the educational content itself and other issues concerning the education of a student in another university are rigidly analysed. As a result, previous study experience of the student can be accepted as satisfying requirements to a part of the degree or its entrance requirements. However, as universities are responsible for the quality and originality of their programmes, the outcome of this process can be negative, that is, no degree or credits can be recognised.

The main international initiative, aimed at the intensification of student mobility and creation of the European Higher Education Area (EHEA), is the BP. The BP provides mechanisms for the harmonisation of educational levels and degrees in different countries and the facilitation of recognition of modules and degrees. For this purpose, three main mechanisms were proposed: a system of HE cycles and National Qualification Frameworks (NQFs), European Credit Transfer and Accumulation System (ECTS), and a learning outcomes-based approach to education [54]. Other educational recommendations and guides (e.g. [57], [67], [58]) propose various organisational measures for the student mobility facilitation, including the harmonisation of mobile students' legal status, and recommended procedures for the students' preparation, integration and re-integration. They call for devising clear recognition policies, expansion of mobility programme initiatives (e.g., Erasmus, DAAD programmes), improvement of awareness about educational systems, legal frameworks and other information that students and institutions need to know about other countries to plan student mobility activities.

All these measures help to some extent to overcome the mentioned difficulties. They form a

normative basis, give direction for developments and stimulate participants towards the intensification of student mobility. However, the role of modern computer technologies in these initiatives is very modest. Their utilisation is foreseen mostly as a supportive tool for the phase when a student is studying according to an existing mobility programme. There is no solid technical solution that would provide support for the process of development of new mobility activities. Nevertheless, such solution could definitely be used to facilitate the fulfilment of the tedious and laborious operations during the authoring of mobility programmes and lead to the intensification of the development of the whole student mobility area.

Based on the presented facts, a technical solution for the student mobility support, with the purpose of student mobility area development facilitation, should be aimed at lifting the information, academic and financial obstacles. It should automate the processing of information for planning student mobility activities, taking into account the peculiarities of educational frameworks and regulations in different countries and universities. Efficient tools for the discovery and analysis of existing educational programmes and student mobility possibilities should be provided, which can relate and compare modules and courses originating from different education providers. Based on this comparison and taking into account regulations and routines specific to these education providers, a track of mobile student, involving recognition of study periods, can be planned. Of course, recognition decisions cannot be taken fully automatically but the results of machine-based analysis can serve as a basis for a more detailed examination by the human expert who takes the final decision. The financial obstacle can partially be overcome since a student mobility support technique can reduce the time that highly qualified experts, engaged in student mobility planning activities, spend on search, processing and analysis of diverse information. Moreover, when using this technique, more options for student mobility activities can be processed and analysed, so the resulting student mobility programme can be planned more carefully, satisfying the strict regulations and various user requirements.

1.2 Problem statement

It is advocated [63, 54, 143, 159] that all student mobility activities, which occur within students' educational pathways in HE, should be well planned and agreed in advance for each individual student. This can guarantee the fulfilment of the agreed plan and raise the quality of the well-planned educational programme. Hence, a technical solution for the student mobility support should explore the problem of planning student mobility activities. For this purpose, in order to designate any educational pathway involving student mobility activities, in this study the notion of a 'programme' is adopted. An Educational Programme (EP) in HE is an approved curriculum route leading to a named academic award that is followed by a registered student [131], [160]. Accordingly, a Combined Educational Programme (CEP) is an approved curriculum route incorporating

student mobility activities that leads to one or several academic awards and is followed by a registered student. If internal student mobility is used, meaning that the programme is taught within one education provider, different courses should be used for the CEP construction. If external or international mobility occurs, different education providers should be involved in the educational process.

In accordance with the aim of a technical solution for the support of student mobility, the main problem solved using this solution is specified as follows. For an individual student or a group of students with similar characteristics, new possible CEPs should be generated based on existing modules and EPs. These CEPs should satisfy the requirements provided by the requester of the programme (an educational organisation or a student) and should conform to the regulations specified by the education providers and the educational authorities. In this study, this problem will be referred as a CEP generation problem.

In the e-Learning field, a similar problem is being solved within Intelligent Tutoring Systems (ITSs). These systems provide a flexible individualised computer-based learning service and carry out functions traditionally appertained to tutors. Curriculum Generation (CG) ITS techniques derive educational curricula for students based on provided educational goals and taking into account students' knowledge and characteristics. For this purpose, these systems should reason about problem domain concepts, corresponding educational goals and available educational resources. To do this, advanced CG techniques exploit planning techniques providing a flexible unified framework for modelling and reasoning about educational activities and their relations with the goals, concepts and resources [170, 174, 163]. Hierarchical planning enhances the curriculum design process with the hierarchical reasoning capabilities [171]. This allows increasing the flexibility and adaptiveness of the CG process, for example, by formalising different tutoring strategies and letting the planner select the strategy for the curriculum development that suits the student the most.

However, within the ITS field the CEP generation problem was not considered before and existing CG approaches do not support the student mobility activities. The problem of their adoption for the CEP generation is particularly interesting, since this extends the CG techniques area of applicability towards the new domain. Moreover, the student mobility area involves issues that are not covered within existing ITS CG techniques. For example, trajectories of students' physical movements should be designed during the CEP development utilising different student mobility scenarios. Requirements to the CEPs can be specified from different perspectives, including requirements to the CEP structure, its official outcomes, physical movements of the student. For the CEP design, modules from different education providers should be extracted and compared. As the CEPs designed should be approved by education experts, a clear and expressive representation for them is needed.

One of the major obstacles limiting the effective utilisation of current CG techniques for the

CEP generation problem is the fact that CEPs are being developed within an environment with heterogeneous regulations. Different educational institutions and authorities have their regulations, established routines and criteria for decision making that should be taken into account during the CEP generation. These regulations manage different aspects of the educational process (e.g., requirements for EPs structure, credit recognition rules, student transfer rules and progression rules) and they are developed and maintained by different people independently. In planning-based CG techniques, these regulations are not supported, and it is assumed that the planning environment is integrally devised by a single author or a group of closely collaborating authors. In order to apply this technology to the CEP generation problem, different authors should have the possibility to contribute to the planning environment specification. Besides, in order to guarantee the smooth planning in such environments and adhere to the division of responsibilities, the process of specification and subsequent planning should be controlled. Scopes where different authors can contribute to the planning environment specification should be limited according to their areas of responsibility and there should be mechanisms to resolve conflicts that can occur between these specifications during the planning. In Information Technologies (IT) field, in order to operate with such heterogeneous regulations, policy-based management technology is used [141, 85]. This technology facilitates management of different types of systems and environments under complex dynamically changing regulations specified by different authors as policies and guarantees that the policies are consistently enforced, that is, the system execution traces satisfy all required policies [27].

1.3 Research questions

Based on the presented motivation and problem statement, the main research question for this study was formulated:

How Combined Educational Programmes (CEPs) can be generated using planning-based techniques in an educational environment with heterogeneous regulations?

Heterogeneous regulations are regulations governing different aspects of the educational process. They are different in different domains of the environment and are authored and supported by different persons independently. As the formulated question is quite general, in order to guide further investigation it was subdivided into more concrete questions:

1. How can planning technologies be adopted to solve the CEP generation problem?
    - Which planning technique is suitable for solving the CEP generation problem?
    - How the CEP generation task can be formulated as a planning problem? Which approach can be adopted for the CEP generation?
    - How can diverse user expectations be specified as input CEP requirements? How can detailed information about the result CEPs be represented to the user?

  - How can different mobility schemas be applied during the planning for the CEP generation and how can composite student mobility scenarios be generated based on these schemas?

2. How can policy-based management approach be utilised to enable planning in environments with heterogeneous regulations?
3. When planning is carried out in an environment with regulations supported by different people independently, how is it possible to have control over the regulations specified by different people and guarantee that they do not contradict with each other and with the general principles of the planning environment designed?
4. Is it possible to improve the planning performance relying on specific characteristics of the CEP development problem area or a planning technique being used to solve the CEP generation problem?

1.4 Research methodology

The overall research approach adopted in this study is constructive research. The distinctive feature of this approach is the construction of a novel artefact that can solve a practical problem in order to elicit new knowledge on how this problem can be solved in principle, and to analyse and compare the solution with other approaches [46]. The choice of constructive research was motivated by the initial problem statement that was focused on the problem solving issues[2] and the absence of existing analogues techniques for solving this problem within the student mobility field. Constructive research is advantageous as by using it a twofold contribution can be achieved: a construct for solving practical problems can be developed (which is usually produced as an abstract model or a framework that can be implemented in different ways) and current knowledge can be extended by studying the construct and the theoretical principles embedded in it. Importantly, the constructive research is a dominant research method in computer science, which this study belongs to. The artefacts produced using the constructive approach in computer science could be algorithms, frameworks, models or languages. It is common that employing the constructive approach as a main method researchers also utilise other research methods to fulfil the local tasks [45]. Now, the main steps of this research are presented and specific tools and methods used at these steps are described.

Step 1. Identification of problem and specification of research question

A problem being explored using the constructive approach should be relevant from both practical and theoretical perspectives [100]. This means that an outcome of study, specifically, a designed artefact, should be aimed at resolution of an actual practical problem and, at the same time, its elaboration should contribute to the theoretical knowledge, for example, by resolving paradoxes or

[2]Based on it, the research question with the corresponding type was formulated, i.e., "How. . . ?".

providing the lacking knowledge. In order to state such a problem and formulate the research question for this study, first of all, the problem domain, the HE student mobility area, was explored. For this purpose, a literature review was done. It covered educational research sources, specifically, journal papers and research reports, and normative and counsel literature, such as actual educational regulations, recommendations and initiatives aimed at the facilitation of student mobility. The case study approach was neglected for this task, as general problems and issues within the area of student mobility should be considered, rather than problems of a concrete university or organisation. Moreover, the case study method would have required substantial time and resources for this step. Additionally, at this stage an overview of corresponding e-Learning and other information systems technologies was done. The outcomes of this step were the problem statement and the research questions specified and a set of requirements to the solution approach from the educational perspective, formulated during the exploration of the problem area (see Chapter 3).

Step 2. Thorough literature review

In this step, the research background was studied through a detailed review of the relevant literature. In the review different directions within the e-Learning field were explored and their possible contributions to the CEP generation were examined. Additionally, the policy-based management was explored as a technique for management of the CEP generation process in the presence of complex heterogeneous regulations. For the searching and gathering of relevant conference and journal papers, digital resources such as IEEE Xplore, ACM Digital Library, CiteSeer and SpringerLink were used.

Step 3. Construction and implementation of solution approach

First of all, as an initial design for the solution a general CEP generation framework was created. The framework defines all components of the solution, including models, technologies and users, their roles and interrelations. Within the framework, the overall CEP generation process was outlined. Next, the refinement of the framework and the detailed CEP generation technique construction was started. As a core component of the developed technical solution, the policy-based planning engine was designed, which joins the strengths of the planning and policy-based management approaches. As is common for the constructive research in the computer science area [45], modelling and formal methods were utilised as auxiliary tools within the design part of this study. The hyper-graph model of the planner's world state and the formal model of the XACML policy language were introduced. They were utilised during the techniques design in order to ease the operation with the corresponding objects, as standard rules and operations can be applied, and guarantee the required properties of the designed techniques. Additionally, as the designed solution employs the planning technique, effectively, it relies on the modelling method. For the policy-based planning technique, a model was specified that provides the means to carry out planning within the student mobility environment and using which the CEP generation task

can be solved. Finally, in this step the working prototype was implemented.

Step 5. Evaluation of solution approach

First of all, the developed approach was evaluated using two case studies. These case studies have demonstrated the feasibility of the approach and provided the insights on its functioning. The case studies were constructed based on different educational scenarios and were different, in scale in order to evaluate the approach in different settings. One of the case studies was constructed with the aim of representing the real life case as close as possible. Secondly, the approach was evaluated based on the specific criteria, which were devised in order to check its required properties and analyse specific behaviour patterns. Finally, experiments were carried out in order to evaluate the planning performance of the designed techniques in planning environments with different characteristics.

Step 6. Deriving conclusions and identification of research contributions

Based on the evaluation of the solution, the conclusions were drawn that describe the achievements attained using the proposed approach and reveal the connections of the approach with existing knowledge. Moreover, a set of directions for the future work were identified, within which more understanding of the topic can be gained and new advances can be reached.

1.5 Scope of the research

According to the problem statement and the research questions posed, in this study we concentrate on the development of a framework and specific techniques within this framework. Using these techniques and framework, the CEP generation problem can be solved, so new CEPs can be generated using existing EPs based on user requirements. This problem was considered more from a technical perspective, rather than an educational one. The educational domain was taken as a problem area for this work. Specific characteristics of the HE domain and, particularly, the student mobility area were explored based on the educational literature and were taken as premises for the technical solution design. One of the most prominent issues that this work focuses on is the existence of heterogeneous educational regulations, which are supported by several authors independently and should be taken into account during the CEP development. For the concrete technique design, within the range of different CEP types we concentrated on credit mobility CEPs as these CEPs explicitly involve credit recognition, reflect the nature of student mobility and are usually prevalent over other mobility types [90].

The educational domain is characterised by a diversity of approaches for the development of EPs, normative requirements specifications and quality metrics definitions. As this work constitutes initial steps in the field of computer technology support for the design of mobility programmes, it accepts an approach based on normative compliance. The technology developed should provide the means for the specification of specific normative requirements regulating the development of

mobility programmes. Correspondingly, the mobility programmes produced by the solution should satisfy these requirements, thereby assuming that they possess the required level of quality inherent in these requirements.

As the core problem to be solved in this thesis concerns a provision of technological support for the educational process (i.e., the CEP development) with the aim of its facilitation, it belongs to the wider e-Learning field. Specifically, the CG techniques exploited in ITSs for solving a problem of non-mobile EP development constituted the required background for our work. Similarly to the advanced CG techniques, we have chosen the planning technologies, in concrete, Hierarchical Task Network (HTN) planning, as the basis for the core CEP generation mechanism design. The planning technologies provide possibilities for reasoning about educational activities and their relations with learning resources and domain concepts. Therefore, the EPs can be developed as detailed networks of educational activities achieving the specified goals. Additionally, the hierarchical planning provides abilities for the hierarchical reasoning and utilisation of pre-built scenarios during this process[3]. The policy-based management approach was explored to solve the problem of handling the heterogeneous regulations during the planning. Finally, the problem of the planning performance improvements was considered in this thesis. In accordance with the central line of our work, it was assumed that the factors of the performance improvements should be based on the peculiarities of the designed policy-based planning technique or specific characteristics of the CEP development problem.

1.6 Success criteria

In order to consider this study a success, first of all, an approach should be proposed that provides means to generate new CEPs using existing EPs and modules relying on planning techniques. The approach should be implemented and its feasibility should be demonstrated using case studies. Additionally, the following specific requirements were stated:

- As the user can have diverse expectations about the CEP that should be developed, it should be possible to specify CEP requirements from different perspectives and on different levels of abstraction.
- The approach proposed should support the reasoning about mobility scenarios and physical tracks that students follow during their education according to CEPs. It should be possible to specify corresponding requirements and generate arbitrary complex student mobility scenarios in order to satisfy them.

Secondly, as part of the proposed planning-based CEP generation approach, a policy-based technique should be designed with which it should be possible to carry out planning in environments

[3]The problem of planning technology utilisation for EP design is considered in more detail in Chapter 2

with heterogeneous regulations. These regulations being specified as policies are to be used to control the planning process. The technique designed should have the following properties:

- Using policies, it should be possible to specify and enforce restrictive regulations, limiting the applicability of actions, and specify established routines that define how certain tasks should be executed.
- In order to facilitate the policy specification, policies should be modular and compositional. It should be possible to specify simple policies independently and unite them using specified rules into more complex policies.
- It should be possible to restrict a set of regulations that can be imposed by different policy authors and define procedures for how different regulations interleave with each other during their evaluation and enforcement.

Finally, one or several planning performance improvement approaches for the developed policy-based planning technique should be proposed based on the specific characteristics of the policy-based planning technique itself or the CEP generation problem considered.

- Experiments should show that these approaches can bring performance gains during the CEP generation.
- Since the performance of the planner depends on the specific characteristics of the planning environment, performance gains produced by the proposed techniques in environments with different characteristics should be evaluated and compared.

1.7 Original contributions

The original contributions of this thesis are as follows:

CEP development framework

First of all, the novel planning- and policy-based CEP generation framework was proposed for the automated development of new CEPs using existing EPs and modules in an environment with heterogeneous regulations. The CEP generation task was not considered before within the e-Learning field. It constitutes the novel approach for the student mobility processes support using computer technologies and is aimed at the student mobility area facilitation.

Policy-based planning technique

The policy-based management approach, which is utilised in different areas of IT, was applied to the new area, the domain-independent planning. As a result, the novel policy-based planning technique was developed. It extends the HTN planning with the possibilities to carry out planning in environments with complex heterogeneous regulations, supported by different people independently. The policy-based planner selects and enforces policies during the planning and guarantees

that the resulting plan conforms with all policies applicable to it. During the development of this technique, the following subsidiary contributions were made:

- Using the XACML policy language, adopted for the specification of policies in the policy-based planning, procedures for the authentication decisions processing and resolution of conflicts between them can be flexibly specified. On the other hand, the procedure of obligations processing during the policy evaluation has not received the due attention. In order to provide the possibility to control which obligations can be produced during the policy evaluation and how they can be jointly executed, the XACML obligations specification mechanism was extended and the **obligations validation approach** was proposed.
- For the XACML policy language there is no mechanism to determine which information about the subject or resource of the policy evaluation request will be required during its evaluation before the evaluation has actually started. Correspondingly, the **adaptive policy requests construction** procedure was designed in order to generate policy evaluation requests containing purposely selected information based on the introduced policies.

Formalisation of the CEP generation problem as a planning task

To solve the CEP generation problem using the policy-based planner, the planning environment describing the student mobility problem area was designed and, in this environment, the corresponding planning task was formalised. Educational processes carried out when a student studies according to a CEP are formalised using HTN constructs and are modelled during the planning, trying different variants of the CEP construction. While a similar approach is used in CG techniques in ITSs, the CEP development problem has a number of distinctive characteristics (see Chapter 2) that were incorporated into the CEP generation planning task specification.

Planning performance improvement techniques

The **postponed policy enforcement** is a problem-independent mechanism extending the policy-based planning technique with the possibility to evaluate policies at earlier stages of the planning. Using this approach, the performance gains are achieved, since dead-ends can be detected earlier during the planning. When not all required information is available for the decision inference, the planner postpones the policy request and re-evaluates it later during the planning.

The **descending policy evaluation** is a performance improvement technique for the CEP generation planning task, which is based on the postponed policy enforcement approach. This technique optimises the process of searching for EPs, which will be used as a basis for the CEP construction, relying on the hierarchical domain structure of the planning environment considered.

Partial policy evaluation for XACML policy language

For the postponed policy enforcement realisation, a mechanism was required for the specification of policy requests containing only some part of the information about the planning action, as well as, an algorithm for such requests evaluation. Therefore, the XACML policy specification language was extended and the partial policy requests specification mechanism with the corresponding policy evaluation algorithm were designed.

1.8 Thesis organisation

The thesis is structured as follows:

Chapter 2 contains a review of existing e-Learning systems and other information technologies that can form a basis for the CEP generation approach development. An analysis is presented of how these technologies can be utilised during the CEP development process support. Among the other technologies, the planning-based CG techniques and the policy-based management approach are considered with special scrutiny.

Chapter 3 contains the student mobility problem domain analysis, based on which the core requirements to the CEP generation system were identified. The CEP generation framework, satisfying these requirements, is presented in this chapter, along with the outline of the general CEP development process.

Chapter 4 contains a description of the XACML policy specification language, which was chosen for the specification of regulations. In this chapter, we also describe the construction of a formal model for the XACML policy language, which is used in Chapter 6 as a basis for the XACML policy language extension.

Chapter 5 presents the design of the problem-independent policy-based planning technique. An overview of the planner's components and the main interaction processes between them, underlying the planner's functioning, are presented. Then, each component is considered in detail; models for specification of the corresponding parts of the planning environment and the algorithms for their processing are developed. As part of the planning technique design, the adaptive policy request construction procedure is introduced.

Chapter 6 describes the postponed policy enforcement mechanism. This mechanism enhances the policy enforcement procedure of the policy-based planning technique and provides the possibilities for the planning performance improvements. For the evaluation of policies during the postponed policy enforcement, the standard XACML policy evaluation procedure was extended and the partial policy evaluation procedure was introduced.

Chapter 7 presents a formalisation of the CEP generation problem as a planning task, being solved by the policy-based planning technique. In addition, this chapter contains a description of the descending policy evaluation technique, which was designed to improve the performance of planning for the CEP development. Specifically, in this technique the postponed policy enforcement

mechanism was applied for the formalised CEP generation planning problem.

Chapter 8 contains a description of the CEP generation system prototype implementation.

Chapter 9 contains an evaluation of the proposed approach and the designed techniques. The overall CEP generation framework is evaluated using two case studies. The policy-based planning technique is analysed against the criteria for planning in environments with heterogeneous regulations. Finally, the performance analysis is done, including the evaluation of performance gains, produced by the descending policy evaluation technique.

Chapter 10 contains general conclusions and a description of possible future work.

Chapter 2

Literature review

Objectives:

- *Review current e-Learning systems and relevant e-Learning technologies that can be used to support the CEP generation process.*
- *Review policy-based management and existing policy languages.*

2.1 Introduction

Student mobility is promoted at national and international levels, while the problem of technological support for the mobility processes by the use of computer technologies has not received the required level of attention. Now there is no system that can provide support for student mobility processes in the automatic generation of CEPs based on existing learning objects (modules and semesters of EPs). The closest area to this problem is the field of e-Learning where diverse technologies for technological support of learning processes are being developed. In this chapter, current prominent e-Learning technologies are reviewed and their correlation and applicability to the CEP generation problem are analysed (see Section 2.2). It was identified that there are techniques within the e-Learning field that can solve a problem similar to the CEP generation, but for non-mobile curricula. They are the Curriculum generation techniques being used in Intelligent Tutoring Systems (ITSs) for the automatic generation of learning paths for students based on their knowledge and individual characteristics. These technologies have a long history and are now successfully used within fast-paced modern web, collaborative and multi-agent e-Learning environments. Among other factors that differentiate the CG and CEP generation tasks, one of the most important is the fact that CEPs should be developed within educational environments with heterogeneous regulations, dictating rules according to which mobility programmes should be built. A policy-based management technology, which is successfully applied in different application areas, can be utilised to overcome this limitation. The policy-based management facilitates management under complex, dynamically changing regulations being specified as policies. Prominent charac-

teristics of the policy-based management and different policy-specification languages are reviewed in Section 2.3.

2.2 e-Learning systems

Computer technologies were used for learning support almost from their beginning. With the development of technologies, their application areas and the value for learning are continually increasing. Due to a variety of e-Learning technologies, many different definitions of e-Learning have been proposed [76]. Researchers who formulate the e-Learning definition narrowly concentrate on the delivering of knowledge by the use of technologies [8, 109] or even on the usage of specific technologies, for example, the internet technologies [7]. In a broad sense (e.g., in [81, 102, 137]), e-Learning is interpreted as a learning or teaching facilitated or supported with the use of information or communication technologies. In [3], it is stated that e-Learning is aimed at the improvement and (or) extension of one or more significant parts of a learning value chain, including management and delivery. Correspondingly, e-Learning technologies include a range of tools for the support of educational activities [105], like learning goals and pathways management, non-digital learning objects management. Further, we review different types of e-Learning systems and analyse the possibilities of adopting e-Learning technologies for the development of a system supporting the CEP generation process.

2.2.1 Virtual Learning Environments (VLEs)

VLE[1] is a type of e-Learning system widely used in HE institutions nowadays [178, 35]. They provide the possibility to author electronic educational content and carry out basic teaching and learning processes using Web technologies with a range of supplementary services, among which the support of 'student-student' and 'student-teacher' communication and the administrative tasks are highlighted as crucial requirements [21, 53]. VLEs are used for pure distance learning, as well as for blended learning[2], supplementing traditional classroom-based learning. Their widespread usage in HE institutions motivates the consideration of VLEs within this study. Examples of modern VLEs are Blackboard, Moodle and Prometheus [104].

The main functions of VLEs are divided into the following areas [31, 104]. *Content development and course design areas* include tools for instructional design, content authoring, sharing and reuse, curriculum management, design of assessments, etc. *Collaboration and communication areas* provide a wide rage of commonly used asynchronous and synchronous means of communication, adapted to educational purposes, as well as specific tools like virtual classrooms, groupwork tools. *Course delivery* functions are used to actually conduct the learning process, providing extended

[1] In different sources, other terms are also used to designate this type of systems, like Learning Management Systems, Learning Content Management System, Course Management Systems.

[2] Blended learning is a mode of study when traditional 'face-to-face' studies are combined with computer-mediated learning activities.

facilities like navigation tools, tracking systems and personal assistants, automated testing and scoring.

Additionally to the two VLE advantages widely referred in the literature [48, 95, 111], that is, a flexible anywhere and anytime learning and a reduction of expenses for teaching[3], well-designed VLEs provide some extra value. They provide additional means to actively involve students in the educational process using new collaboration and communication tools and, hence, to change their learning mode from passive to active one [21, 79, 180]. Actively using these tools within the course building schemas that provoke students collaboration, for example, granting to students some teachers functions, a VLE can become a virtual space where students and teachers share their knowledge, resources and ideas. For tutors, tracking, assessment and reporting tools of VLE are the sources of information that can be used as a basis for the analysis and corresponding revision of the educational process: course content improvement, changing of the presentation mode for a student, planning future educational courses for specific skills and competences building [53, 107]. Finally, as the course content is specified in a modular and formalised way, it can be actively re-used in different courses and individual learning paths can be easily built for students.

Despite the fact that VLEs are widely used in universities, the possibilities of their utilisation for the development of a system supporting the CEP generation process are restricted. VLEs are focused more on the inner content of the modules and corresponding processes. They lack full-fledged support of EP curricula, while information about these significantly facilitates the CEP development process and is needed for a CEP generation system. Moreover, learning path construction and adaptation, when it is required, should be done manually in the VLE by tutors [68]. There are initiatives [5, 108] where VLEs are extended with the adaptation technologies, but they lie more in the ITS field, which will be considered further (see Section 2.2.2). Actually, if distant or blended learning modes are used, university VLEs can be used as distant learning platforms for the CEPs developed by a CEP generation system. Then, the virtual learning mobility can be realised, that is, when students study according to a CEP, they can be (simultaneously) enrolled and take part in learning activities in VLEs of different universities participating in the CEP. Moreover, VLEs can be used as sources of information for the CEP development, as they are widely used in universities and store valuable information about their educational modules (like credit values, pre-requisites, learning outcomes, etc.) However, as their specifications lack a fixed standard structure and often use different units and terms, the utilisation of this information is currently difficult and requires extra unification and integration efforts[4]. Therefore, the integration with VLEs and their usage in a CEP generation system both as sources of information about modules and as target

[3]These advantages actually correlate with the advantages of the distance learning.

[4]While IEEE Learning Objects Metadata standard [97] is supported by some VLEs, it cannot represent all required information for the CEP generation, for example, credit values for modules. So, for harvesting module information using this standard, a specialised profile based on this standard should be developed.

distant learning platforms can be considered as an attractive but non-core functionality for a CEP generation and support solution.

2.2.2 Intelligent tutoring systems (ITSs)

ITSs are tutoring systems providing flexible individualised learning using AI technologies [24]. These technologies are utilised to carry out tasks that traditionally pertain to tutors and help to create systems, "*which know what they teach, who they teach and how to teach it*" [123], p. 252. Historically, ITSs appeared as an extension of Computer-Aided Instruction (CAI) systems, which were earlier ancestors of VLEs [71]. When web technologies appeared, they were recognised as a favourable platform for the ITSs, so they started to be actively used for the development of ITSs [125]. With the migration to the new platform, ITS techniques were also enriched with novel presentation and navigation methods that originally appeared in the web-technologies field [23].

ITSs were developed for a variety of domains with a focus on different educational tasks. Model-tracing ITSs are used in problem-solving environments. They are designed to model correct courses of action for solving specific types of problems. When the student deviates from the correct solution track, this is detected and the system provides a valuable feedback. For example, the model-tracing PACT Geometry ITS forms an intellectual environment where students can solve geometry tasks. This ITS was developed as part of school geometry course [4]. Simulation-based ITSs are developed for simulation environments, where students can carry out exercises. These ITSs coach students by providing tutorials, giving tasks using the simulation environment and assessing the results. For example, AIS-IFT system was developed to carry out ITS functions for a helicopter pilots simulation environment [133]. Collaborative ITSs carry out tutoring tasks in virtual collaborative environments where a group of students is working on some task. ITS can guide the collaborative decision process, propose peers for consultancy, promote collaborations. In the area of medicine, the COMET system was developed for the collaborative diagnosis of disease under the control of an artificial tutor [149].

2.2.2.1 ITS architecture

The ITS architecture consists of four main modules: student module, domain module, tutoring module and communication module [71, 123]. Each of these modules contains a corresponding model, using which the knowledge required for the functioning of the module is specified.

The *domain module* is responsible for storage and processing knowledge about the problem domain that should be taught. Depending on the type of ITS, the knowledge stored in the domain model is different in nature and has different levels of granularity. In most cases, based on the domain model, a student's cognitive state should be built and further tasks for the student should be selected. For this purpose, the domain model should contain descriptive knowledge about the domain: concepts, skills and techniques existing within the domain and their relationships. In

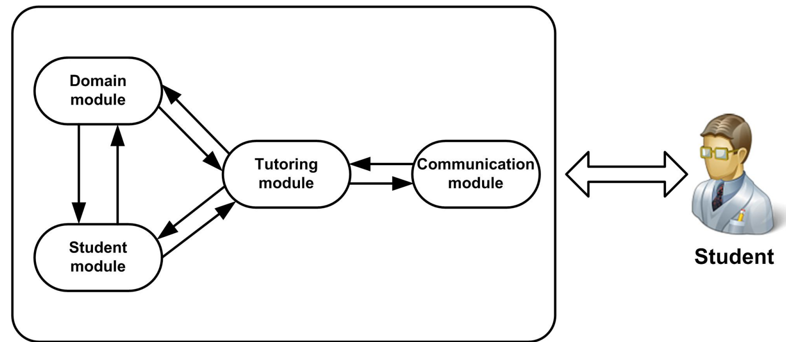

Figure 2.1: Generalised models and functions of ITS

problem-solving ITS, the domain model should additionally contain procedural and explanatory knowledge, so that a solution proposed by a student can be checked and informative feedback can be given. In model-tracing systems, this expert knowledge is encoded using production rules representing the course of problem solving. At a lower level of granularity, they can even represent atomic operations that can be carried out by a student when he (or she) solves the problem. In constraint-based systems, the domain model is formed by constraints defining states which incorrect students' solutions can occupy. Domain models of the simulation-based ITSs should additionally contain information about properties and behaviours for each type of components within the simulation environment.

In ITSs, learning materials are presented to the students using specially designed digital learning objects. These learning objects are usually stored in a repository within the domain module. Metadata of the learning objects contain their characteristics, required during the instructional planning and, actually, the tutoring. These characteristics can contain their roles in the tutoring process (theory, exercise, hint, bibliography, example, etc.), media types (text, picture, film, simulation problem, test, etc.), levels of complexity and interactivity, and other properties. In order to plan the tutoring process, which consists of the interaction of the student with the corresponding learning objects, the learning objects should also be linked with the concepts stored in the domain model. In addition to the domain model, the domain module should also have mechanisms using which the domain model can be processed and information required by the student and tutoring modules can be retrieved from it and provided to these modules.

The *student module* serves two main purposes: it stores all information about the student and, based on this information, it infers new knowledge about him (or her). This module receives information about all activities carried out by the student in the system and, supported with information from the domain model, dynamically maintains the student model.

The student model is divided into low-level observations of the student activities, his (or her) knowledge model and information about his (or her) learning style. A history of the student's

interactions with the system is stored in the observation model. Depending on the type of ITS, this model can refer to the learning objects and/or problems studied by the student, his (or her) assessments results or lower level information (e.g., the number of times a specific page was opened). In the model-tracing ITSs, this model also contains sequences of operations that were carried out by students during the problem solving. The student's knowledge model is usually populated based on the lower-level observations. Often, it is built using the domain model of the ITS. For example, the student's knowledge model can be defined as a subset of the domain model. Using this model, it is possible to determine the level of mastery for every unit of the domain model. Such a model is called an overlay model. For the specification of these models, probabilities or fuzzy values can be used [28]. Information about the student's learning style can contain conclusions about the type of tutoring recommended to the student, including the level of interactivity, the format (text or graphical) of learning objects, the approach (inductive, deductive), the semantic density value, etc. This information can be derived before the tutoring using special tests, which the student should pass [126], or it can be updated based on the observations of his (or her) progress during the tutoring.

The information from the student model, as well as the domain model, is utilised by the *tutoring module* to manage the core tutoring process within the ITS. First of all, the tutoring module, based on the information about student's knowledge, determines next topic that should be taught to the student. When the topic is chosen, it determines an educational activity which should be carried out (e.g., assessment, theory lesson, revision) and chooses a specific learning object which will be used. During the learning, the tutoring module guides the student and provides him (or her) with feedback. For example, in the problem-solving ITSs, it evaluates the solutions proposed by the student or informs him (or her) when a wrong operation is performed during the problem solving. When a student navigates through the material himself (or herself), the tutoring module can provide hints or guide the navigation [164]. For these operations, the tutoring module should make many decisions, influencing the performance of the learning: which topic to choose, which activities and in what order to use, learning objects with which properties to select, how much to intervene into the learning process, how detailed hints to provide, etc. First of all, the tutoring module should be adaptable: all these decisions should be taken based on the information from the student model. The tutoring module can adapt the difficulty of the learning objects selected, the nature and order of the activities proposed to the student based on the proficiency and preferences of the student. Moreover, based on the derived characteristics of the student's learning style, the ITS can vary the learning strategy used. Advanced ITSs store the supported learning strategies as separate pedagogy models and change them to adapt to the student.

The *communication module* is responsible for all interactions with the student, including screens layout, dialog management, tracking student behaviour.

2.2.2.2 ITS main techniques

The major techniques used in the ITS field are as follows [24, 23]:

- **Curriculum generation** is used "*to provide the student with the most suitable individually planned sequence of knowledge units to learn and sequence of learning tasks (examples, questions, problems, etc.) to work with*" [179], p. 372. Using this technique, the ITS defines or maintains an optimal curriculum for the student.
- **Intelligent analysis of student's solution** techniques are used to analyse solutions proposed by the student. Problems can vary from a simple multiple choice test up to a complex construction task. In addition to the correctness analysis, these techniques can specify which component was incorrectly used or constructed, provide some explanations or specify an area of the student's domain knowledge that should be improved.
- **Interactive problem solving support** techniques are used to control the student's problem solving process. They can detect deviations from the correct solution construction path, notify the student about it and provide assistance (refer to the theory, give hints, correct the solution, etc).
- **Adaptive presentation** techniques originated from adaptive hypermedia systems. In this technology, course pages are generated or assembled from pieces of content for a specific student based on his (or her) knowledge, learning goals and specific characteristics.
- **Adaptive navigation** techniques are used to adapt links shown on a web page for the current student, in order to help him (or her) in navigation and orientation.
- **Adaptive information filtering** is used to select from a large repository few educational units which fit the student's request most.
- **Intelligent collaborative learning** is a group of techniques used to control and promote a collaborative learning in ITSs. This group includes adaptive group formation, adaptive peers choice, collaboration support and virtual students techniques.

As the problem of the CEP construction is a special case of the curriculum development problem being solved using the CG techniques, next we will concentrate on this functionality of ITSs.

2.2.2.3 Curriculum generation techniques

Curriculum generation (CG) is the core functionality in ITSs. The CG techniques can be used to structure educational material at different levels within ITS. At a high level, it is used to create sequences of topics that the student should study. At a lower level, it structures learning objects that are used to teach these topics. As a basis for the development of individualised learning paths for students, different information can be used. Usually, the current student's domain knowledge is used. Some systems also take into account different properties of the student, like the preferred type of the learning content or medium. Advanced CG techniques based on the information about

the student select the tutoring strategy appropriate for the creation of the student's curriculum [71]. There are two different types of CG techniques: active and passive [23]. When the passive technique is used, by default the student navigates through a standard course. A system intervenes only when the student makes a mistake or explicitly asks for assistance. On the other hand, the active type proactively generates a whole course for the student or at each step determines the learning object that should be taught next. Obviously, for the CEP generation task the active CG techniques should be considered.

Different methods were used for the implementation of CG functionality. In general, they can be classified into graph-based techniques and planning-based techniques. Next, these techniques are illustrated using examples of concrete ITS systems.

ABITS (Agent Based Intelligent Tutoring System) [28] is an agent-based ITS that was developed to extend the functionality of traditional VLEs and make the learning process more personalised and adaptable using the student modelling and CG functions. The domain model is represented in ABITS as a conceptual graph where nodes are concepts of the underlying knowledge domain and edges are relations between them. Three type of relations are supported: prerequisites, sub-concepts and a general relation. Learning objects in ABITS are atomic web-deliverable resources that deliver lessons, tests and simulations. For storing information about learning objects, the IEEE Learning Object Metadata (IEEE LOM) standard is used [97]. One of its data elements is used to map the learning object to a set of concepts within the conceptual graph that this learning object refers to. The student model consists of the student's cognitive state and preferences. In Figure 2.2 relations between the constructs of the student model and other models are presented. The cognitive state is an overlay of the conceptual graph, where the level of confidence that the student knows specific concepts is represented by fuzzy numbers. The student's preferences are also stored as a set of fuzzy numbers: each number represents the level of confidence that the student prefers learning objects with specific value of the meta-data attribute (e.g., specific format, level of interactivity).

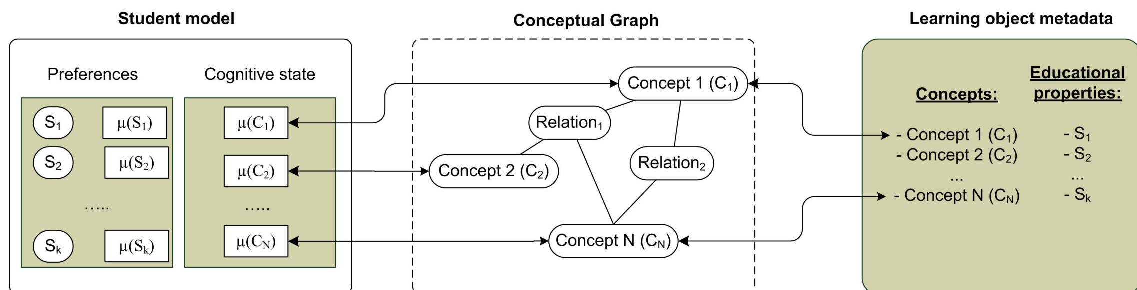

Figure 2.2: Conceptual graph, student and learning objects models of ABITS

The automatic CG is carried out in ABITS based on the student model and the learning goal

that should be achieved. The learning goal is a set of concepts within the conceptual graph. In the first step, the system finds a list of concepts which should be studied by the student in order to know all concepts within the learning goal. For this purpose, the depth-first tree traversal is carried out in the sub-graph formed by the prerequisites relations of the conceptual graph. When a concept that the student knows is reached, the traversal is stopped. Next, this list of concepts is transformed into a sequence of learning objects explaining these concepts. A learning objects is selected for teaching a concept based on the student's preferences. Finally, required test units are added into the resulting sequence of learning objects.

The graph-based approach to the curriculum sequencing is widely used. For this approach, it is crucial how the domain and learning objects models are defined: which relations they contain and which of them are utilised for sequencing [12]. The CG algorithm in ABITS is a basic algorithm where prerequisites between concepts and student's preferences are utilised for the concepts sequencing and learning objects selection. The common alternative is an approach where prerequisite relations are defined between the learning objects and this graph is utilised for the sequencing (e.g., in [89]). In the concept graph, other types of relations can be used during traversing, like 'subconcept' or 'similarity' [172]. Another enhancement used in CG techniques is the utilisation of traversing algorithms constructing optimal paths based on the defined criteria. For example, in [88], the joint ordered graph containing both concepts and learning objects is weighted: each learning object node is assigned with an average time required for its studying. Then, using the graph traversal algorithm, a path with the minimal duration is constructed. Generally, graph-based approaches are simple and elegant but they lack mechanisms for the representation of different knowledge that can be used during the CG. As a consequence, their flexibility and adaptability is less than for other approaches. In the next paragraphs, an ITS system is presented where the graph-traversing CG algorithm was extended using planning- and rules-based approaches.

The **DCG (Dynamic Courseware Generation)** approach, proposed in [25, 171, 172], is targeted at the development of individualised courses for students based on their knowledge and characteristics. The domain structure in DCG is represented as an AND/OR graph: nodes represent domain concepts, edges represent different semantic relations between them. In this domain structure, different types of relations are used, so the same domain can be represented from different perspectives. In addition to common 'prerequisite' and 'aggregation' relations, special types of relations can be used to represent different aspects of notions under the consideration, for example, their physical or functional organisation. Each concept corresponds to a set of teaching materials that can be used to teach it. The teaching materials, in addition to properties like media type and complexity, are characterised by their roles in the pedagogical process, for example, introduction, motivation, example, explanation, analogy or test. The student model is also constructed as an overlay model, but in DCG it is based on the probabilistic theory.

The DCG is a combined CG technique that uses graph-based, rule-based and planning techniques. Based on the learning goal specified, that is, a set of concepts that should be taught, the DCG initiates a bread-first traversal of the domain structure AND/OR graph. At each iteration, so-called discourse rules are used to control the traversal. The discourse rules are production rules that define criteria according to which learning materials should be structured for students with specific characteristics. They define which type of semantic links should be chosen to follow, and when it is required to switch to another type of semantic links. For example, for an intelligent student a top-down approach to the discourse can be used, so the traversal should follow refinement links in the concept structure, while for other students the bottom-up approach can be used. The outcome of the graph traversal is a concepts plan for learning (i.e., a partially ordered set of concepts).

In the next phase, a teaching strategy is chosen. The teaching strategy is an approach which should be used to teach a concept to a student. The strategy can be unstructured, meaning that a student can select which task will be carried out next, or structured, when the student is led by the system. The structured strategy determines a specific pattern of teaching tasks that should be sequentially carried out to teach the concept to the student, for example, make an introduction, present theory, refer to examples and pass tests. Special strategy-selection rules are used to decide which strategy should be used for a student based on his (or her) characteristics and knowledge. Each strategy is encoded as an AND/OR tree. Nodes of this tree are teaching tasks, AND and OR links designate alternative decompositions of these tasks to lower level teaching tasks. During the curriculum construction, specially specified rules are used to choose between alternative AND-decompositions in the strategy and to select teaching materials which should be used for the execution of teaching tasks. These rules are correspondingly referred to as teaching methods and teaching material selection rules. The resultant sequence of leaf tasks forms a course plan that should be carried out for teaching the current student. This plan is passed to the executor module that communicates with the student and performs the teaching tasks.

Extension of the graph-traversal CG approach with the rules- and plan-based techniques in the DCG led to the construction of a CG system with advanced adaptation possibilities. An approach for the concepts plan development, a teaching strategy and methods for the execution of teaching tasks are selected for a specific student using different types of expert rules. The decomposition of educational tasks using AND/OR graphs is analogous to the hierarchical planning. This approach is beneficial as all knowledge that is used at different levels of the curriculum design is formalised and stored in a specialised knowledge base. So it is possible to easily modify and extend the curriculum design knowledge, and to choose opportune curriculum construction methods from this knowledge base during the educational programme development for a specific student.

TOBIE (Teaching Operators-Based Instructional Environment) is an ITS that was

designed with a focus on CG. This functionality in TOBIE is fully implemented using a unified planning mechanism [173, 170, 174]. The domain knowledge base for TOBIE is represented as a directed AND/OR graph similar to one in the DCG. It represents a decomposition structure with different types of semantic links. The domain knowledge base contains different types of knowledge, stored at different levels, for example, skills and goals decomposition levels, conceptual level, problem-solving level. The domain knowledge base also stores pedagogic knowledge, for example, possible curricula specified on the domain concepts level. All knowledge in TOBIE, including the domain knowledge base, is formalised in a unified and modular way using Teaching Operators (TOs). A TO is a construct, similar to planning operator, that encodes possible transitions of the student model using preconditions and effects. When a TO is executed, the corresponding predefined teaching procedure is carried out and the student model is updated according to the effects part of the TO. TOs can also encode hierarchical structures: within the teaching procedures, lower level TOs can be referred. Pedagogical knowledge about the context of TO usage is also encoded within the TO structure. Preconditions of TO define when its execution is appropriate, relying on available knowledge about the student, for example, his (or her) characteristics and knowledge. A diagnosis part of TO refers to several remedial operators and a diagnosis operator used for evaluation of student's response during the TO execution. When the diagnosis operator detects an exception, the remedial operators are used to identify an error and carry out the corresponding remedial procedures.

The instructional planning mechanism in TOBIE provides designers with a large flexibility in the ITS construction, since in TOBIE unified constructs can be used to define different aspects of the educational process. However, on the other hand, different types of knowledge are specified in a mixed form in the knowledge base and cannot be isolated. For example, TOs are specified in a way that the pedagogic knowledge is merged with the knowledge about organisation of concrete domain. So, when a new ITS is built, the whole knowledge base should be designed from scratch. Generally, TOBIE illustrates how planning mechanisms can be utilised at different levels and for different aspects of the educational programme development.

Analysis of CG techniques towards the CEP generation problem

Pure graph-based techniques are not appropriate for solving the CEP generation problem because in the student mobility area an overall graph connecting different concepts and learning objects cannot be built. Learning objects for the CEP development (EPs and their modules) and specifications of domain concepts are provided by different universities. Relations between them, for example, the prerequisite relations, are usually defined by the content designers only within a specific university. Relations between learning objects and concepts used in different universities can be derived using knowledge-based approaches, but only in a non-binary form as a degree of confidence that the relation exists or as a measure of the relation strength. Such relations would

connect a learning object with all other learning objects, so decisions that should be taken during the CEP construction cannot rely on the absence or presence of the relation, as it is usually done in the graph-based approaches. More complex threshold mechanisms should be exploited where a concrete threshold value to use depends on a type of decision being taken (e.g., a recognition or a prerequisites evaluation) and on local regulations influencing this decision (e.g., how similar two learning objects should be in order for one of them can be considered as equivalent with another).

Rule-based and planning-based techniques are more appropriate for the CEP generation task as they are not rigidly tied to graph structures. Moreover, they can support different externally specified educational strategies that can be utilised for the CEP construction. An appropriate strategy can be selected based on a current situation during the planning. It is advantageous to use a planning approach as a basis for the CEP construction because, as opposed to rules, planning employs action-based approaches with the explicit ordering and time support. This makes it possible to naturally represent an EP and simulate educational processes using planning models with a required level of detail. Rule-based approaches are usually used as supplementary mechanisms to facilitate planning and graph-based techniques. Rules are used to explicitly specify principles for taking specific decisions during the CEP construction and, hence, make corresponding processes easily extensible, more intuitive and adaptable. The majority of planning-based CG systems supports hierarchical planning, so in the next section we will concentrate on this type of planning.

In current CG techniques, the following issues were found that limit their applicability to the CEP generation problem:

- Current systems are rigidly designed by one author or a group of colleagues collaboratively. They lack a support for regulations that are different in different domains (e.g., countries, universities, faculties) and which are specified and supported by different persons independently. These regulations include laws, legal acts, established routines and expert assessments, which are in force only in a specific region and can be applicable only in specific cases. Such regulations manage various aspects of the educational process, like the structure and content of curricula, admission rules, student transfer routines, rules governing student progression, rules for prerequisites evaluation. In spite of the fact that current CG techniques actually provide the possibility to specify different types of knowledge that can be utilised during CG, it is assumed that a knowledge base with a specific type of knowledge is specified exclusively by one person.
- In current ITSs, as an input for the curriculum construction, a set of concepts to study is specified. When a CEP is developed for a student in a real HE environment, this is not enough. Two more requirements are critically important: award the student will get at the end of the education and the structure of the developed CEP. Structure requirements could

determine universities where the student will study, and how many and which transfers he (or she) will make during the education.

- The problem domain in current ITSs is defined using concepts and relations between them. Learning outcomes are usually not supported, what contradicts with the BP requirements. Learning outcomes contain more information than a list of concepts a student should know. They can be requirements for the focus of education (learn theory, learn how to apply it, have an overview, etc.), the level of cognition and other aspects of the student achievements. Similarly, current ITSs lack support for other mechanisms proposed in the BP for the comparison of learning objects originating from different sources: educational credits and frameworks for qualifications [54].
- Finally, current ITSs cannot support environments where different terms and unit are adopted in different domains for description of the same or related notions. For example, these terms and units can be used to describe learning objects that should be used for the CG, so a transformation mechanism for them is required.

2.2.3 Planning technologies for Curriculum generation

AI planning technology is useful for generating a plan consisting of actions that achieve a specified goal state, given a description of the initial state and a formal description of the domain. So AI planning is a constructive technology that has been applied in many different areas, including electricity networks, spacecraft mission control, manufacturing, web-services, robotics, evacuation and unmanned vehicles control.

The theoretical model that underlies classical planning is the action-based state transition model of a system defined as $Sys = \langle S, A, \gamma \rangle$. S is a set of all system states, A is a set of possible actions. The state-transition (partial) function $\gamma : S \times A \rightarrow S$ defines a state where the system transfers from a current state when an action is carried out. Commonly, a set of restrictions is applied when a model of the system is specified for planning. It is assumed that the system's model is finite (contains a finite number of states and actions), static (is changed only by actions), deterministic (γ is a partial function) and fully observable (state s is known fully) [116]. In classical representation, the system's state is represented as a set of ground positive function-free first-order literals $l(t_1, \ldots, t_n)$ under the closed-world assumption (this state is also referred to as a planner's world state). Actions are represented using operator schemas. The operator schema contains a precondition, which is a first-order formula that should be true in a world state before the execution of operator, and an effect defining modifications of the world state that should be carried out as a result of the operator execution. Effects are represented as sets of positive and negative literals[5]. A plan is a sequence of actions that should be executed from the initial state in order to reach a goal state.

[5]Positive literals are added to the world state, negative literals are removed from it.

2.2.3.1 Hierarchical Task Network (HTN) planning

Classical AI planners carry out a trial-end-error choice of actions in order to find a sequence of actions leading to a goal state. This approach results in a need to explore an excessively large search space (although heuristics can make the planning more efficient). Prominent examples of classical planners are STRIPS [61], UCPOP [127], FF [75]. One of the approaches proposed in order to solve this problem of classical planners is the HTN planning that utilises domain-specific knowledge in order to guide the planning process and explore only the parts of the search space that could lead to a correct plan construction. In HTN planning, a planning problem, in addition to an initial planner's world state, is specified as a set of tasks. These tasks are abstract representations of activities that should be carried out by the planner. 'Methods' (or refinements), which are specified by a domain author, are used to represent alternative schemas for the execution of compound tasks. They are used to decompose compound tasks into networks of lower level tasks and eventually into actions that are executed in the planner's world state in an ordinal manner using operators. This approach is different from classical planning in terms of how a planning goal is specified and how the planning is carried out. Using HTN planning, a user can get more control over the planning process by the specification of initial network consisting of compound tasks. It was shown that HTN planning is strictly more expressive than the classical planning [26]. In HTN, it is possible to specify problems that cannot be specified in classical planning. However, the main advantage of HTN planning is a considerable reduction of the search space that a planner should explore using methods, representing knowledge on how a solution should be built. Due to these advantages, HTN planners are more widely used to solve real-world planning problems than other types of AI planners [116].

Original ideas of HTN planning were proposed in the NONLIN [155] and NOAH [136] planners. These ideas were applied and extended in more modern HTN planners described next. O-Plan2 [154, 37] is a domain-independent general planning and control framework with the ability to utilise diverse domain knowledge. The focus of the O-Plan2 development was on the extensible architecture that provides means to support and facilitate interactions between task specification, planning and execution components. O-Plan2 unites different AI techniques for planning, including HTN planning, agenda-based approach, least commitment, different constraints handling techniques (including temporal and resource constraints). The agenda-based approach resembles a blackboard system, where a set of 'issues' represents outstanding decisions or requirements that should be resolved. At each planning cycle, the system opportunistically chooses which issue type and which specific issue should be resolved and calls a Knowledge Source, which possess corresponding processing capabilities in order to make decisions and modify a current plan. Reasoning over different types of constraints results in possibilities to prune the search space. The generic

architecture of O-Plan2 provides the means to construct different planners, combining different Knowledge Sources, constraints managers and other components, and facilitates the support and extension of the resulting planning systems. O-Plan2 was applied to a wide variety of application domains, including construction and house building, logistics, crisis response, evacuation operations and many others.

SIPE-2 [183] is a domain-independent hierarchical, non-linear planner, which was designed for solving practical problems. It gives much consideration for planning efficiency and reacting to events that occur during the execution of constructed plan. During the planning, SIPE-2 memorises possible choices and uses a notion of context in order to determine a branch that should be followed at this point. When an event is produced during the execution, the planner can react to it initiating re-planning using an alternative branch that involves minimal modifications of the current plan. Other SIPE-2 possibilities include reasoning about resource and time constraints, intermingle planning and execution. The planning algorithm of SIPE-2 is a depth-first backtrack search. In order to achieve heuristic adequacy and produce results to a large set of planning problems, it utilises different heuristics that limit the search space and reduce the complexity of the planning. Other features of SIPE-2 include a GUI for interaction with the user during the planning. SIPE-2 was utilised as a basis for the development of a distributed multi-agent planning system [44]. Also SIPE-2 was used as a planning engine in the integrated planning environment Cypress that includes a planning domain specification and storage platform, a reactive execution system with dynamic re-planning features and a reasoning engine with the uncertainty support. Examples of SIPE-2 practical applications include air campaigns, military operations, production line scheduling, construction planning.

SHOP [119] and SHOP2 [117] are domain-independent HTN planners that utilise a forward tasks decomposition. They decompose tasks and apply operators in the same order as they will be carried out at the execution stage. This strategy is attractive as at each stage of the planning the current planner's world state is fully specified. This provides the means to evaluate complex conditions to prune irrelevant plans in the current planner's world state, and call external functions in order to carry out complex domain-specific computations based on information extracted from the current world state. SHOP supports only fully ordered task networks in the current task network and in decomposition methods. In SHOP2, this limitation was relaxed. While it plans only in a forward manner, its current set of tasks can be only partially ordered. Hence, as SHOP2 supports interleaving of tasks that has been produced as a result of different compound tasks decomposition, some planning domains can be specified in SHOP2 in a more concise and efficient form. The described characteristics of SHOP2 led to a distinguished performance award in the International Planning Competition. SHOP and SHOP2 were applied to a large number of practical problems, including web-services composition, evacuation planning, project planning and many

others.

2.2.3.2 Planning technologies for Curriculum generation and Combined Educational Programmes development

There are a number of examples of planning technologies application to the CG problem [12, 13, 162, 163, 170, 173, 174], some of which were described before. The majority of planning formalisms used for CG supports hierarchical tasks, because they naturally represent educational processes and provide additional control of the planning process for a domain author. Moreover, the hierarchical planning provides gains in the planning performance due to the controlled exploration of the planner's search space with the help of methods used for compound tasks decomposition. In simple, non-hierarchical planning-based CG systems (e.g., [13]), a straightforward approach is used. Courses are represented as operators. The preconditions specify course prerequisites, while the effects define the learning outcomes of the courses. This approach is characterised by a limited expressivity and poor scalability, in comparison with more advanced approaches where learning objects are used as action and task parameters [163]. In the later approaches, when a learning object is added to the EP being developed, its role within this programme is specified explicitly. It is determined by the action that introduces this learning object into the educational plan. This provides the means to add extra meaning into the resulting educational plan and construct more complex EPs[6]. Moreover, this approach gives the possibility to store learning objects externally in a repository, rather than in the planner's world state [162]. At the level of compound tasks, which can be decomposed using methods, a similar differentiation of approaches exists. On the one hand, more abstract compound tasks and decomposition methods are used to formalise domain-independent knowledge about the teaching methodologies and routines. On the other hand, planning constructs analogous to the HTN tasks and methods are used to specify decomposition of concepts that can be utilised to build up a curriculum or problem-solving methodologies that can be applied only in specific problem domains (e.g., integration by parts in integration calculus). A clear advantage of the former case is a possibility to re-use these teaching strategies for different problem areas, but a concrete approach should be chosen based on the purpose of the system and the level of granularity at which the hierarchical planning will be applied. For a CEP generation system which is not tied to a specific domain area and should operate on the level of complete HE curricula, the former case, where domain-independent abstract tasks and methods are used, is more appropriate. Moreover, this approach is advantageous for the CEP development as using HTN methods different EP development routines can be specified.

Current instructional planners tend to use ordered tasks decompositions, where tasks are de-

[6] For example, learning objects can have several modes of usage that are activated based on its role in the plan, e.g., the same testing unit used as an intermediate task during the course and as a final test task can apply different criteria of assessment.

composed in the same order as they will be fulfilled during the education. As educational goals are strongly inter-dependent, ordering relations between these goals provide valuable information during the planning, which determines the planner's decisions [169]. An optimal choice of the current action (or current learning object) during the education is possible only when the history of the student's education is known. Based on this history, the student's current knowledge can be determined and his (or her) preferences and characteristics can be inferred. Moreover, the teaching strategies and EP development routines defined in a domain-independent manner cannot be specified without information about the ordering between the tasks that form the strategy.

Several issues arise when planning technologies are applied to the CG problem. As the nature of the education is non-deterministic and a student's progression depends on his (or her) outcomes, much efforts have been spent on the development of reactive instructional planning approaches [172, 169]. In such a system, a curriculum is built in advance but during the educational process the system 'senses' the environment and can react appropriately (e.g., this approach was implemented in TOBIE [173]). Another challenge, which has not attracted much attention, is the requirement to present extended information about the curriculum development process and the curriculum structure rationale to the user. This is required when the resulting curriculum is returned to the user, who should be able to fully understand the principles of its development and augment it when this is required. For example, in [162, 163], for these purposes it was advocated that not only the development process, but the final curriculum itself should be hierarchically structured.

The main drawback of current planning technologies for their application to the CEP generation problem is the assumption that a single author or a group of closely collaborating colleagues is responsible for the planning environment specification. The environment consists of methods and operators, which determine possible actions that could be carried out in the environment and limitations over them. Relying on this specification, an automated planner develops a plan. In a CEP generation system that should consider heterogeneous regulations, specified by different authors, this assumption is not fulfilled. For a feasible solution to the CEP generation problem using planning technologies, it is required that several independent authors should have the possibility to contribute to the planning domain. Each of them has its own area of responsibility, but as they can determine requirements for the same process from different perspectives or represent authorities at different levels responsible for the same process, their areas of responsibility can overlap. In terms of the planning environment constructs, they can determine conditions on the execution of specific operator or task and determine task structures produced during a task decomposition in a specific situation. Additionally, it should be controlled how these authors can influence on the planning process. It should be possible to specify a set of situations where a specific author can contribute and control outcomes of his (or her) intervention into the planning process: the overall feasibility of the planning process should be preserved and possible conflicts between contributions

of different authors should be resolved. A straightforward approach when each author has a set of operators and methods, over which he (or she) has full control, does not provide the required level of control, as outcomes of these operators and methods execution are directly applied in the planning environment. Moreover, this approach leads to a growth in the number of operators and methods, as each author should have a distinct set of operators and methods that he (or she) can specify.

In order to fulfil these requirements, which would enable planning in environments with heterogeneous regulations, a specialised mechanism operating on top of the described planning technologies and extending their functionality is required. Within the field of automated planning, there are two areas where the related problems are being solved: distributed planning and planning under control rules.

Control rules-based planners apply the same general approach for the planning efficiency improvement as the HTN planners. Specifically, domain-independent planning techniques are extended with the possibility to encode domain-specific knowledge that can be utilised during the planning in order to guide the planning process and, thereby, improve the planning efficiency [116]. In the control rules-based planners, such domain-dependent knowledge is specified as heuristic rules. These rules are used by the planners in order to make decisions during the planning which affect the planning efficiency and for which other criteria cannot be efficiently used. It is commonly alleged that the control rules are a special form of the planning algorithm (strategy) specification [10]. Control rules are used to prune parts of the planner's search space that do not contain correct plans or contain only less desirable plans (for example, longer or more expensive plans).

Control rules specified as production rules are used in the PRODIGY planner [29] to guide the search process with the aim of planning time reduction and improvement of the quality of plans. The PRODIGY planner exploits two planning mechanisms during the planning, namely, state-space forward search and partial order backward-chaining. Three types of control rules are used in PRODIGY: reject rule, select rules and prefer rule. Correspondingly, using them it is possible to prune from the search space one branch or all branches except the selected branch or give priority to one of the branches. The effects of the rules specify which method of progression should be used (forward or backward), which goal should be achieved next, which operator should be selected, which object should be used to instantiate a variable, and so on. Within the condition part of the rules, information about the current planner's world state, achieved and unachieved goals, and other meta-level information based on previous decisions taken during the planning (e.g., selected or applied operators) can be used. When several control rules are applicable during the planning, they all should be enforced during the decision taking.

TLPlan [9, 10] is a control rules-based planner where control rules are specified using first-order version of Linear Temporal Logic (LTL). In contrast with PRODIGY, this planner utilises

only forward-chaining search. In TLPlan, control rules are specified as logical formulae involving temporal operators that should be true in all states within the planner's world states sequence corresponding to the produced plan. Correspondingly, control rules can make reference only to information within the planner's world state (and, using specially added 'goal modality', to the planner's goal). So, in comparison with PRODIGY, control rules cannot explicitly refer to actions that should be applied and objects that should be used for the variables' instantiation in operator schemas (this limits the possibilities for specification of regulations considerably [11, 70]). The control rules in TLPlan are checked in an incremental way against the sequence of states corresponding to the prefix of the plan being generated. When it is detected that the control strategy is violated, the current branch is pruned from the planner's search space. Thus, in order to compose the control knowledge in TLPlan, it is not necessary to know details about the planning mechanism or planning actions specified. Control knowledge is defined as part of the domain description and determine the properties of this domain that should be satisfied in order to efficiently achieve the planning goals.

The development the next control rules-based planner, TALplanner, was inspired by TLPlan [96]. It also employs only the forward chaining search principle and utilises temporal logic formulae for the specification of control rules. TALplanner is based on Temporal Action Logic (TAL), a narrative-based linear discrete metric time logic, which is used for reasoning about actions and changes. In contrast with TLPlan, the evaluation process of the control rules specified in TAL is optimised using the pre-processing technique [47]. Different sets of optimised control rules are produced that should be evaluated only after the corresponding operators. These control rules take into account information about the operator executed and, hence, can be evaluated more efficiently. Additionally, the specification of operator is also updated by moving some conditions for the evaluation of control rules into the operator's precondition.

Other planners that employ control rules for the planning efficiency improvement or as a tool for the specification of additional constraints on the plan are based on one of the two approaches that were introduced in the PRODIGY planner (e.g., [106, 98]) and in TLPlan/TALplanner (e.g., [51, 52]). Control rules specified as production rules have a simple and comprehensible structure and within their effect parts planning actions can be referred to explicitly. As a result, for this type of control rules-based planners, techniques of control rules generation are created using machine learning methods [106, 98]. On the other hand, control rules specified using temporal logic are expressive in using temporal modalities. Modern planners usually utilise compilation approaches, similar with one used in TALplanner, in order to enforce this type of control rules. For this purpose, corresponding restrictions on their syntax are adopted [51, 69]. Such control rules are converted into finite automata that should be simulated during the planning. This provides the means to evaluate the plan validity based on the control rules. For the specification and simulation

of these automata, specifications of the operators are extended with corresponding conditions and automata states updates.

In spite of the fact that control rules are employed to guarantee that the planning process conforms with the specified requirements, adoption of the described approaches for the specification and enforcement of educational regulations during the planning for the CEP development has the following issues. It is supposed that control rules, as well as the core part of the planning environment, are specified by a single author or a group of collaborating authors, so there is no mechanism for independent specification of packages with control rules by different authors (within their areas of responsibility) and their consistent joint enforcement during the planning according to the purposely defined procedures. When several control rules should be enforced together, either all of them should be enforced simultaneously (however, this is not always required) or the approach employing the static numeric priorities is used (however, this approach is not scalable and flexible). The widely used compilation approach according to which the control rules and the domain specification should be transformed into a new domain specification, where these control rules are always enforced, makes the control rules specification inflexible and prevents their timely updates.

Distributed planning considers problems when several independent problem solvers are building plans that should be executed within the same environment concurrently or even constitute parts of a single plan achieving the overall goal. The distributed planning mechanisms should guarantee that these plans can be executed in a coordinated manner. For this purpose, at a minimum, no conflicts should arise during their execution and, preferably, the overall utility should be maximised [181]. Therefore, the distributed planning system should produce plans consistent with each other and achieving the overall goal or (and) the individual planners' goals. For this purpose, different specialised coordination mechanisms were designed aimed at the resolution of specific types of conflicts and interactions during the planning. These mechanisms depend on the type of environment considered and include plan merging, pre-planning conflict resolution, negotiation and other approaches [43]. In contrast with the centralised planning, a planning environment for distributed planning can be specified by several authors independently, but only in parts where possibilities and preferences of different problem solvers are specified [44]. Correspondingly, the issue of independent specification of different regulations that control the planning process being carried out by a single problem solver and their consistent enforcement, which is important for the CEP generation, is not within the scope of issues considered in the distributed planning area.

Generally, distributed planning does not concentrate on the issue of flexible specification of different heterogeneous regulations that should be taken into account for solving problems within some specific planning environment, especially, when these regulations can be updated dynamically. Rather, it provides the means for resolution of specific types of conflicts and interactions that

arise in some sorts of planning environments using coordination mechanisms that constitute an integral part of the automated planning system. Distributed planning techniques, in addition to the automated planning field, are actively being developed within the multi-agent systems (MASs) area of research. These systems are also widely used in e-Learning and their applications are described in the next section.

2.2.4 Multi-agent e-Learning systems (MASs)

A MAS is a system composed of several autonomous agents, functioning continuously within the environment and communicating with each other either directly, or through the environment [140, 181]. MAS technologies provide the means to construct a complex system as a collection of several autonomous agents, which can play different roles and perform different functions, and exploit higher level interaction schemas to define advanced patterns of their interactions. Using this approach, tasks within complex, open and dynamic (unpredictable) environments can be effectively solved [82]. The multi-agent approach is advocated for systems that possess some of the following properties: data, control or expertise are distributed, high flexibility or interoperability is required, there is a need to concurrently achieve multiple goals or there are multiple methods to solve the problem [152, 110, 83].

The multi-agent approach is actively exploited for the implementation of distributed e-Learning environments and integration of existing Learning Objects Repositories (LORs) and e-Learning systems (e.g., VLEs) for joining the communities and optimisation of educationalists efforts. The agent paradigm is used for wrapping existing e-Learning systems and organisation of cooperative service provision (e.g., learning object search service [14, 124]). MAS technologies are used for management of heterogeneous and dynamic e-Learning environments where it is required to adapt to changes in a timely manner and guarantee the quality of service. The multi-agent approach provides a required level of scalability for these environments, and flexibility and adaptability for their control. Autonomous agents ‘sense’ changes in the environment and optimise service parameters, foreseeing future changes and adapting them based on known user characteristics (e.g., this approach is used for management of mobile e-Learning services [147, 148]).

Existing MAS-based e-Learning environments do not support student mobility and do not provide the CEP generation functions. In some e-Learning systems, the core tutoring process, including the generation of curricula, is implemented based on the multi-agent approach (e.g., in ABITS [28], described in Section 2.2.2.3, and other systems [175, 126]). However, the functionality provided by these systems is equivalent to the already described CG functionality. The main advantages of this paradigm are seen when it is necessary to actively interact with the student and generate curricula dynamically, monitoring the student’s actions and reacting to them. In general, the utilisation of MAS technologies for the implementation of an e-Learning system with the focus

on the instructional planning, including the CEP generation functionality, is justified when data, control or expertise should be distributed in it. For the CEP generation process, this depends on the chosen pattern of interactions within the system, that is, if local LORs are used, where modules and EP metadata are maintained, or if universities are ready to pass their learning objects to a central repository, where these learning objects will be utilised for the CEP development, along with the formalised rules and regulations of their usage, and others issues.

2.3 Policy-based management

Policy-based management is a flexible and effective tool for the information systems management. Policy-based management is usually used for systems that have difficulties with their control due to the specific characteristics of these systems or specific requirements for their management. Such system characteristics can be the distributed nature, autonomy or heterogeneity. Control of such systems can be difficult, for example, due to the complex rules that these system should conform to. Another factor, leading to difficulties in the system control, is the dynamic modifications of these rules and their heterogeneity, meaning that they can define different aspects of the system behaviour and can come from different sources. Such difficulties arise in many information technology domains, like networks, distributed systems, pervasive computing and web services, so different approaches within the policy-based management field were developed to resolve these difficulties. Some vendors of network hardware (e.g., Cisco, HP) use policy languages as a standard tool to control their equipment. In this section, the notion of policy is examined and a classification of policy rules is provided. Next, we will analyse different policy languages and approaches and choose a language that can be used to satisfy the requirements specified for a CEP generation system.

2.3.1 Policy definition

In the information technology the most general and well known definition of policy was given by Morris Sloman:

> "Policies are rules governing the choices in behaviour of a system" [39], p.1.

In this definition two things are implied. First of all, the possible behaviour of system is more general and policies limit possible traces of the system. Secondly, policies should be somehow enforced for this system, voluntarily or by force. In the following definition, these two aspects are explicitly stated and more details are provided:

> "Policy can be defined as an enforceable, well-specified constraint on the performance of a machine-executable action by a subject in a given situation" [20], p.368.

In [20], the following clarifications were also given for terms used in this definition:

- Enforceable: using the system infrastructure, it should be possible to sense and control (i.e., to prevent or enable) the execution of actions controlled by the policy.

- Well-specified: policies are well-defined declarative descriptions.
- Machine-executable action: policies can control not only actions being entirely executed within a machine environment, but also actions that can be executed outside the machine environment if afterwards the fact of their execution is reported to the machine.
- Subject: the subject can be a human, a hardware or a software component (or a collection of such entities).
- Situation: the applicability of policy to a current situation is limited by the policy preconditions and a variety of contextual factors.

A system usually has more possible traces than permitted by the policy, so the policy constrains the system behaviour [182]. A mechanism that guarantees that the behaviour of the system satisfies the policies specified for it is called an enforcement mechanism. An enforcement mechanism depends on the type of system and on a policy language used for the specification of policies. Policy enforcement is described in Section 2.3.4.

Policies specify only information aspects of the desired behaviour and have declarative semantics. They describe which requirements the system traces should satisfy, without specification of full action sequences that must be carried out by the system. Due to this property and due to the presence of enforcement mechanisms, which mediate in interactions between the policies and the system itself, policies can be separated from the core part of the system. Hence, the policies can be flexibly changed in a dynamic manner, without the need to modify the system itself (e.g., recompile it).

Another important property attributed to policies is persistency [39]. Policies do not represent actions that are fulfilled only once during the system execution. Instead, the policy governs the behaviour of the system at present, as well as in the future.

2.3.2 Benefits of the policy-based management

The main benefits of the policy-based management are as follows:

Facilitation of system management under complex regulations. Complex regulations are specified naturally as a set of short declarative rules applicable only in specific situations (e.g., people tend to formulate regulations as rules) [18]. Policies specified as sets of such rules in turn can be combined to form more complex policies using composition mechanisms that define principles of their interaction and joint enforcement within a single policy. These policies are automatically enforced using enforcement mechanisms that, among other things, track the interactions of policies and apply pre-specified conflicts resolution rules [19]. Additionally, some policy specification languages have mechanisms that provide the possibility to specify policies at a more abstract level, for example, specify policies applicable in specific situation or for specific group of entities. Moreover, for some policy languages additional tools

exist for the simplification of policy specification task. They can be specialised GUI for policy specification [38], policy testing and validation tools [62], policy specification techniques for non-technical users [144], and others.

Context sensitivity. The enforcement mechanism should select and enforce policies which are applicable to the current situation. Different policies can be enforced depending on the time, location, role of the user, etc. [120]

Support for dynamically changing regulations. Policies can be changed when the system is running without the need to stop it. These changes of policies can be carried out manually by the administrator or they can be programmed on a specific time or event [80].

Regulations reusability. Policies are declarative representations of behavioural constraints. So they can be saved, archived and reused, when it is required, during the operation of the system [141].

Regulations can be supported by different persons independently. Different policies can be specified and updated independently by different persons, as composition mechanisms are used to define rules for their combination into an overall policy. Moreover, such policies even can be stored in a distributed manner and collected only for evaluation (if this is required) [99].

Explicit license for autonomous behaviour. Policies are an appropriate tool for the implementation of selective control. Policies provide specific instructions for some situations, for instance, prescribe to execute some action or avoid it. Therefore, the choice of specific decision that will be carried out, provided that this decision is conformant with the policies, is at the discretion of the system itself [168].

2.3.3 Types of policy rules

Policies are utilised for the management of different systems and, moreover, are used to control different aspects of their behaviour, such as access control, administration and interactions. In spite of this fact, policy languages usually adopt a common general schema for the policy specification where policies are specified as a set of simpler building blocks that jointly specify the system behaviour. These building blocks are usually called rules or primitive policies. The types of rules that are most often used in policy languages are presented in Figure 2.3. This schema was developed as a result of the review of different policy languages and is based on the results of previous policy rules analysis attempts [42, 30, 39]. All rules at the schema are divided into complex and simple rules. As opposed to the simple rules, explicit references to other rules can be utilised within the specification of complex rules. For example, the delegation rules can define who and under which conditions can delegate rights granted by specific authorisation rules.

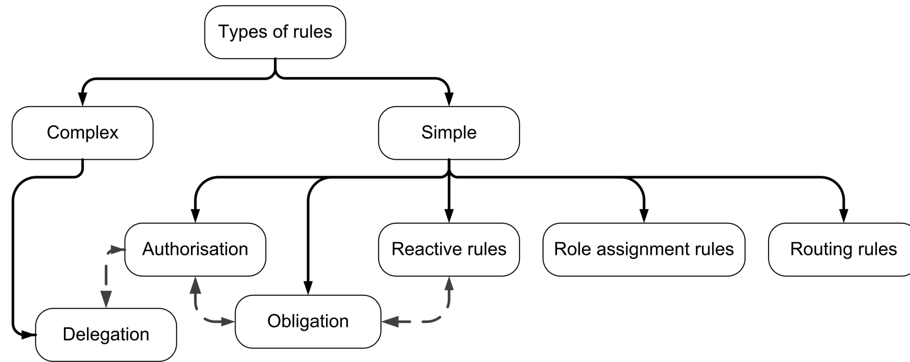

Figure 2.3: Types of policy rules

2.3.3.1 Authorisation rules

Authorisation rules define which operations can be executed by a user or other active entity (e.g., an agent). Positive authorisations define a set of permitted operations; negative authorisations define a set of denied operations. Authorisations are usually defined in the form '*subject - action - resource - condition*'. Subject is a set of active entities that this policy is applicable to. Action defines operations permitted (or denied) for the execution by the corresponding subject. Resource defines a target object for the operation, that is, a resource being accessed. The condition part can include arbitrary constraints that define when this rule should be enforced. The following is an example of an authorisation rule:

"*Permit Bob (Subject) to Log-in (Action) to SystemX(Resource) from 9 a.m. till 6 p.m.(Condition)*"

Authorisation rules are commonly used in access control policy languages (e.g., XACML [153], Ponder [41]). Other types of policy languages can also utilise rules similar to the authorisation rules that define if specific actions can be carried out. For example, in interactions management policy languages [114], such rules can specify if a specific action can be carried out over a message by a controller (e.g., forward or deliver).

2.3.3.2 Reactive rules

Reactive rules define the behaviour of a management system, which can be monolithic, like an administration tool, or distributed and heterogeneous, like a set of administrative agents that control different equipment throughout the network. These rules are specified using the form '*event - condition - action*' (ECA). So they define which action(s) a system should carry out in response to an event occurred within the managed system. The condition part is optional. The policy author can use it to specify constraints on a current state that should be satisfied in order to trigger the rule. This type of rule is used in different policy languages, but it is usually attributed

to the management policy languages. For example, in an administration system it can be required to send an e-mail to the administrator if a severe virus infection is detected. This can be formalised using the following ECA rule:

"*On System infection (Event), If Virus is severe (Condition), Do Send e-mail (Action)*"

ECA rules can also be used to schedule the execution of some maintenance actions, for example, deep antivirus scanning.

For the implementation of these rules, a set of pre-defined events within a system should be defined, on which the evaluation of conditions is initiated. Such events can be time-based or action-based (e.g., a specific packet arrival). Using this type of policy rules, the Internet Engineering Task Force (IETF) policy model was defined [39].

2.3.3.3 Role-assignment rules

Role-assignment rules define requirements that a user should satisfy in order to assign him (or her) a specific role. Role-assignment rules are usually used as part of a role-based access control mechanism implemented using a role-based policy language (e.g., TPL [42]). The evaluation of these rules for a user is initiated when he (or she) requests access to a system. Usually the role-assignment rules are used at the server side, where users are not known in advance and in order get an access to resources a user should possess specific certificates. Role assignment rules are specified in the form '*condition - role assignment*'. The condition part is used to check if a user possesses the required certificates and the role assignment part is used to specify which roles should be assigned to the user, if the condition is satisfied. Based on the roles, assigned according to the role-assignment rules, an authorisation mechanism should determine specific access rights for the user.

2.3.3.4 Routing rules

Routing rules are a dedicated type of rules used in network routing policy languages. Routing rules define permitted traffic routes in a network, based on the known sender and receiver of a packet and other parameters, like the type of a packet or current time. Path-based routing rules are a special type of routing rules that were proposed in PPL (Path-based Policy Language) [146]. These rules are more expressive than ordinal routine rules. In addition to a sender and receiver, explicitly specified path-patterns are used in these rules in order to determine their applicability. It should be noted that the routing rules are usually rigidly tied to a specific application domain within a network management or routing field and are not meant for other applications.

2.3.3.5 Delegation rules

Delegation rules are used to define which operations can be delegated from one subject to another. A delegation rule specifies a set of subjects who can delegate their rights, which rights can be

delegated and to which subjects these rights can be delegated. The access right being delegated is usually defined using constructs similar to authorisation rules. Delegation rules can have the following form '*Subject* - *Grantee* - *Resource* - *Action* - *Condition*' that defines that *Subject* can grant to *Grantee* a right to carry out *Action* on *Resource* and *Grantee* can carry out this action only if *Condition* is satisfied. Delegation rules are used in access control policy languages, in addition to the authorisation rules (e.g., in Ponder [41]). Actually, delegation rules are a special type of authorisation that defines when a subject is authorised to execute a special delegation action, leading to changes in access rights for other subjects.

2.3.3.6 Obligation rules support

Obligation rules specify which actions should be performed when certain events occur. Obligations are used in different types of policy languages, from security policy languages to management policy languages. Several different methods exist for the specification of obligation rules. The utilisation of a specific method depends on a type of policy language where the obligation is used.

Obligation rules can be classified into obligation rules associated to ECA reactive rules and obligation rules associated to authorisation rules. The former obligation rules have structures similar to ECA rules. Correspondingly, they can be triggered by any event which can occur in the system. A distinct element of this rule is the subject part. A policy author using the subject part can specify which component or user is responsible for the execution of the triggered action. Enforcement mechanisms used for this type of obligation can also be divided into two classes. In the first class, an action triggered by an obligation rule is transferred to a component that is a constituent of the management (administration) system. This component, being a part of the enforcement mechanism, must eventually execute this action. For example, in the Ponder language these components are automated managers, deployed in the environment [49]. In the second class, subjects of obligation rules are users or autonomous entities. They are informed about the obligations triggered and should execute them, but, at the same time, they can refrain from their execution. Such obligations are used in the KAoS policy language, where obligations are passed for execution to autonomous agents [168]. The following is an example of such obligation rule:

"*UserAgent (Subject) must Notify user (Action), when Contact list member goes on-line (Event), if time is from 9 a.m. till 9 p.m. (Condition)*"

As this obligation should be executed by a personal agent, it can refrain to show this notification, for example, if it considers that the user is busy at that moment.

Obligations associated to authorisation are specified and enforced along with authorisation rules. In this group of obligation rules, a special sub-group can be distinguished. This sub-group contains obligations that should be transformed into authorisation rules for their enforcement.

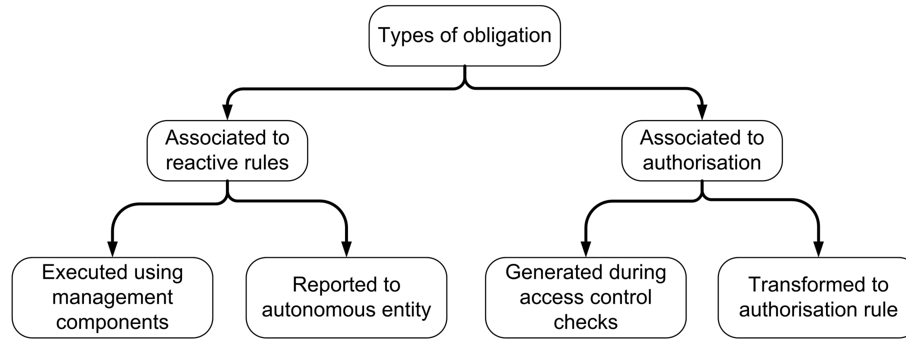

Figure 2.4: Obligation types

Such obligations with the form '*Subject S must do action A on event E*' are transformed into authorisation rules that take authorisation decisions based on information about the history of actions execution: "*Subject S cannot do B, if event E has occurred and S has not done A*". Another example of such obligation is a conditional for of obligation used in the Security Policy Language (SPL) [134]: "*Subject* S_1 *must do action* A_1 *if subject* S_2 *has done action* A_2". For enforcement, this rule is transformed into a policy with a dependency on a future event: "*Subject* S_2 *cannot do* A_2*, if subject* S_1 *will not do action* A_1". Such rule is enforceable in SPL using the security monitors that support transactional autonomicity[7].

Another type of obligations associated to authorisation are obligations specified as additional actions related to an authorisation rule or an authorisation policy. These obligation actions should be triggered when corresponding authorisation rules (or policies) are enforced during the authorisation checks. An example of such a rule is:

"*Deny Bob (Subject) to Read (Action) FolderX (Resource) and Make log entry (Obligation)*"

which states that when Bob's access to FolderX is denied, an enforcement mechanism should make the corresponding record in a log file. Such obligations are implemented in the policy languages XACML and EPAL [153, 129]. Obviously, the former class of obligations is just another form of access control rules specification, while the latter class brings additional possibilities for the policy specification and enforcement.

2.3.3.7 Rule types analysis

In this section, the described policy rule types are analysed and compared for specification and enforcement of regulations during the planning-based CG. As was shown in Section 2.2.3, a core element of planning-based approaches is an action. Actions are used as basic constructs for modelling a planning domain using action languages. Correspondingly, actions are the main elements which a planner reasons about. The planner chooses actions and organises them into a plan being

[7]Such security monitors can reject actions based on information about events that occur after their execution, but within the same transaction.

produced as a result of the planning. Hence, for a policy language that is used for the management of planning-based CG process, it is advantageous to have action-based rule types. Such rule types are authorisation, obligation, reactive rule and delegation.

When there is a need to specify educational regulations that impose specific constraints on a CEP being developed or on an educational process carried out when a student is studying according to a CEP, authorisation languages can be used, considering that the CEP development process is implemented out using a planning system. In classical representation for planning problems, every modification of a system state should be modelled by an action. Hence, in order to have control over the planning environment and all processes being modelled using it, these constraints can be specified on actions that actually lead to the planner's state updates. For example, constraints on a possible track of a mobile student can be represented using constraints on actions carried out when this track is modelled during the planning. Authorisation rules provide the means to specify different constraints based on fine-grained attributes, which describe the action itself or represent other important information.

Educational regulations also often define routines that should be executed in order to carry out a specific task or prescribe which actions should be executed in specific situations. For example, these routines can specify a process that should be carried out when a mobile student joins a university. Policy rules that can prescribe execution of actions are reactive rules and two types of obligation rules. Implementation of reactive rules and obligations associated with them will require implementation of a dedicated mechanism that will monitor the system state and generate pre-specified events that can trigger the policy rules. However, first of all, in an action-based system planning, state modifications can be done only as a result of the action execution. Secondly, in classical representation for planning problems, an action's effect is fully known before its execution has started[8]. So the implementation of additional mechanisms for state monitoring is not required and the evaluation and triggering of obligations can be united with the selection of actions and the evaluation of authorisation rules for them. Delegation rules are not relevant for our problem, as delegation is out of the scope of a planning environment that models the CEP development and only core educational processes. Moreover, delegation rules can be specified as a special type of authorisation rules.

2.3.4 Policy enforcement

A policy enforcement mechanism guarantees that the system behaviour conforms with the policies specified. There are two main approaches for implementation of this mechanism: outsourced and provisional approaches [30]. Correspondingly, the main difference between these approaches is in a component that receives information about the current situation, analyses policies and generates

[8]In classical representation for planning problems, the planning environment is static (does not model external events) and deterministic [116].

a policy decision.

The outsourced approach refers to the policy enforcement mechanism which was proposed as the ISO/IEC 10181-3:1996 [78] standard. Its schema is presented in Figure 2.5, A. The Policy Enforcement Point (PEP) detects requests to resources and submits corresponding requests to the Policy Decision Point (PDP) for the policy evaluation. When a policy decision is returned from the PDP, it enforces it, meaning that it rejects or permits the request. PEP is application-specific and, in fact, can constitute a part of the application. PDP is application-independent. It carries out the role of a global policy engine that receives decision requests from PEPs, retrieves the information needed for their evaluation, carried out the evaluation based on the specified policies and generates policy decisions. Therefore, the decision taking functionality was moved from the PEP, which actually enforces decisions, so this approach is called outsourced. It was implemented in such languages as XACML [153] and EPAL [129].

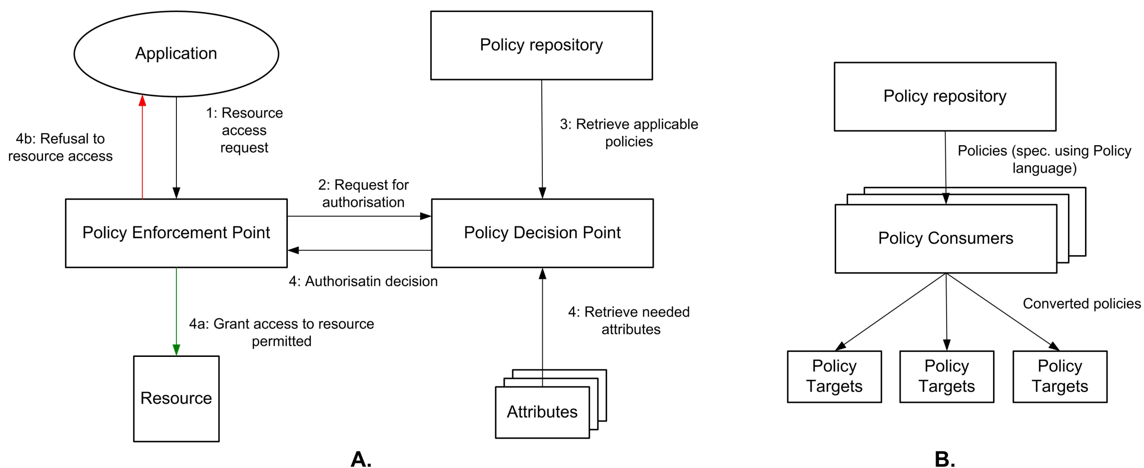

Figure 2.5: Outsourced (A.) [6], p. 54 and provisional (B.) [103], p. 6 policy enforcement models

The provisional policy enforcement approach was specified within the IETF Policy management system requirements model [103][9]. Its schema is presented in Figure 2.5, B. This approach is based on two main types of components: application independent Policy Consumers (PCs) and application (device) specific Policy Targets (PTs). In this approach, PCs only retrieve policies from the policy repository and distribute them to corresponding PTs, which actually enforce them in the system. As PTs are application specific, policies should be converted to an application-specific format at the PT or PC side. This approach was implemented, for example, for the Ponder policy language [38]. Authorisation policies specified in this language are compiled and deployed as access control policies for Windows stations or as firewall rules.

[9] The IETF Policy management requirements model contains specifications of both policy enforcement approaches.

2.3.5 Policy composition

In large and distributed systems, like modern enterprises, policies are usually specified and updated by several persons independently [101]. For some policy languages (e.g., XACML), advanced techniques were developed to combine policies from different sources and enforce them in a coordinated manner. These techniques include policy groups, scoping and policy composition and involve conflicts resolution [184]. A basic mechanism that, in general, facilitates the policy administration and, in concrete, can be utilised for specification of policies originating from different sources is policy groups. A policy group unites a set of related policies, which, for example, can refer to the same domain of the environment or determine a specific aspect of the behaviour. A widely used approach for the realisation of policy grouping is to allow policies to be arbitrary nested. In this case, composite policies can be created as groups of lower level policies and form a hierarchical policy structure. The support for hierarchical policies is advantageous as it provides the means to structure complex policies and forms the basis for the introduction of policy composition operations. This approach was used in XACML [153] and SPL [134]. In the Ponder language, several types of composite policies, which can be nested, were introduced, for example, groups, roles and relationships [40].

In order to enforce policy groups that originate from different sources in a coordinated way, a policy language should provide the possibility to specify when a specific policy group should be enforced and how to resolve conflicts between different policy groups if they arise during the policy evaluation. To this end, a policy composition is used: a set of policy composition operations (algorithms) is defined that contain operations used to form higher level policies out of lower level policies and specify relations between them. Composition operations define how the composed policies will be processed during the policy evaluation. Composition operations can be used to determine which policies are applicable and should be evaluated in a specific situation and how a final decision for a composed policy should be inferred based on the decisions produced by the evaluated policies. For example, using a *scoping*, it can be specified for which situations a policy can be applied (as it was defined in [19]). During the policy evaluation, a policy is processed only if the corresponding scope expression is satisfied. The scoping is useful when it is externalised from the policy or policy group specification. Then, the scoping can be used to manage policies received from different sources: the policy applicability can be controlled without the need to modify the policy specification itself. For this purpose, an independent scoping operation can be defined or a scoping expression can be used as a construct within a policy group, thus limiting the applicability of the constituent policies (e.g., this approach is used in XACML [153]). Using composition operations, conflict resolution strategies between policies can be defined. For example, if authorisation policies A, B and C are united into a policy group using some composition operation, this composition

operation can impose one of the following conflict resolution strategies. If some of policies A, B or C return positive authorisation decisions and some of them - negative decisions, a positive decision overrides strategy states that the action will be authorised by the policy group (as at least one policy authorises the action). On the contrary, if a negative policy decision overrides strategy is used, the action will not be authorised (according to it, all policies should produce only positive decisions). A policy composition operation can impose a specific order between the policies: if a policy returns a positive or negative authorisation decision and policies that have higher priority have not returned any decision, this decision should be taken as a policy group decision. Alternatively, more complex or application specific conflict resolution strategies can be defined.

Accordingly, policy composition provides the means to construct complex hierarchical composite polices using a modular approach simplifying the policy specification and providing the possibility to enforce policies from different sources. Composition operations are used to define relations between policies and policy groups that can be specified by different authors. The policy composition functionality is necessary for the specification of educational regulations governing the CEP development. These regulations are specified by different parties and at different levels of the educational system independently (e.g., programme leaders, faculties, universities, states, etc.) So there is a need to combine these regulations and enforce them during the CEP generation in a coordinated manner.

2.3.6 Policy languages

In this section, existing policy languages are compared. For the comparison, we have selected policy languages that support authorisation and obligation rules and for which application-independent policy enforcement engines exist.

2.3.6.1 Ponder

The Ponder [41, 40] policy language was designed as a universal security and management policy language with the focus on the distributed systems and networks management. It supports positive and negative authorisations, obligations and delegation policies. For the specification of a managed environment and grouping elements within it (e.g., subjects, resources and organisational units), Ponder supports domain hierarchies. Accordingly, subjects and resources that a policy is applicable to are specified as domains within these hierarchies. A policy is applicable to all objects that are descendants of the domains referred to as subjects/resources in this policy. For grouping policies, the Ponder language supports several types of composite policies. They are roles, which unite policies with a common subject; relationships, which unite policies managing rights and duties of roles towards each other; management structures, which represent organisational units and contain roles, relationships and other management structures; and policy groups, which can unite

any policies. Composite policies in Ponder can be nested and form hierarchies.

Ponder is an object-oriented declarative policy language supporting inheritance and policy typing that is useful for the management of policy specifications and their re-use. Policies specified using the Ponder language are compiled and deployed in the managed infrastructure. Authorisation policies can be propagated to corresponding infrastructure components as firewall rules or access control policies for Windows stations. Obligations in Ponder are specified as ECA rules. They are passed to corresponding automated agents that carry out the administrative actions. These agents receive events, generated by the event service, and use ECA obligation rules stored in their knowledge bases in order to infer actions that should be executed in response to the events.

2.3.6.2 KAoS

The KAoS policy service and domain services [167, 150] were developed for enabling policy enforcement for agent frameworks (it was used in Nomads, Voyager and other frameworks). They were also adapted and successfully applied to general Web services environments [168]. KAoS can run on heterogeneous environments; it supports dynamic policy updates and includes advanced administration tools. The KAoS domain services provide the possibility to structure objects like software components, people, resources and policies to facilitate the deployment and support of policies. KAoS policies are specified based on the KAoS policy ontologies. These ontologies are used to represent actions, actors, groups, locations and policies themselves. KAoS supports four types of policies: positive/negative authorisations and positive/negative obligations. Interrelated policies can be combined into policy sets. The main component of a policy is an action class. It contains constraints on the action parameters (including action's subject, target, relations with other actions) that determine the applicability of the policy. Obligations in KAoS are defined as an action that should be executed by an autonomous agent relatively to the occurrence of some event. The events in KAoS can be start or end points of the action execution. The specification of polices as ontologies provides the means to use corresponding reasoning capabilities, for example, for the policy analysis and design-time conflicts detection and resolution.

Enforcement mechanisms for KAoS policies are flexible and support both outsourcing and provisioning models. The implementation of a concrete model depends on the infrastructure used. For example, for obligations two types of enforcers exist: monitors and enables. Monitors and enforces observe the execution of obligations. When a monitor detects that an agent has not fulfilled the obligation, it can apply a sanction to the agent. On the other hand, enablers in such situation try to carry out the action on behalf of the agent.

2.3.6.3 EPAL

The Enterprise Privacy Authorisation Language (EPAL) [129] was developed by IBM and was targeted at the specification of enterprise-internal privacy policies that control data-handling within

enterprise information systems. EPAL policies and policy requests are specified as XML documents using defined schemas. They abstract from application-specific details like a data model or an authentication schema. EPAL supports positive and negative authorisation rules and obligations. EPAL authorisation rules, in addition to subject, resource and action attributes, contain a 'purpose' section. In this section, the purpose for which access is requested should be specified. EPAL policies use hierarchical categories to specify subject, resources, actions and purposes. This approach is similar to domains structures in Ponder and provides the possibility to propagate policy rules in a hierarchy. Rules in an EPAL policy are evaluated according to the specified precedence order and decision from the first applicable rule is returned. Obligations in EPAL are specified as part of the authorisation rules. When an authorisation rule is enforced, actions specified in its obligation section should be performed by the enforcement mechanism.

IBM submitted EPAL version 1.2 to the W3C for consideration as a privacy policy language standard.

2.3.6.4 XACML

The eXtensible Access Control Markup Language (XACML) [153] is targeted at the specification and enforcement of access control policies within an enterprise in an application-independent manner. It supports positive and negative authorisation rules and obligations. Authorisation rules are specified as conditions on attributes of the subject, resource, action and environment, under which an access can be granted or denied. These conditions are divided into pre-requisite part (target), which has a restricted form and is used to select policies applicable to a policy request, and a condition itself, which can be represented as an arbitrary complex condition expression. Obligations, similarly as in EPAL, are defined as part of the authorisation rules and should be executed by the enforcement mechanism when the authorisation rule is enforced. XACML expressions support different data types and functions. The content of a resource can be represented as an XML document within the policy request and used as a source of information to infer a policy decision.

XACML policies can be defined in a modular manner. Policies consist of policy rules. Policies themselves are composed into policy sets, which, in turn, can be arbitrarily nested. Component policies can be distributed throughout the network, as nested policies can be addressed using references and retrieved for the evaluation when it is required. For the coherent composition of policy rules and policies in XACML, a set of combining algorithms are used that define different routines for component policies and rules processing during the policy evaluation and conflicts resolution strategies applied at run-time (e.g., rules/policies precedence, modality precedence, single policy/rule strategy). XACML is an extensible XML-based language. For its extension, standard extension points are defined in the XACML specification [153]: for new combining algorithms, new

data-types and functions.

XACML versions 1.0, 2.0 and 3.0 were approved as OASIS (Organisation for the Advancement of Structured Information Standards) standards [50][10].

2.3.6.5 Policy languages comparison

Policy languages described in the previous sections were compared against the requirements for a policy language that could be utilised for the educational regulations specification used for management of a planning-based CEP generation process. These requirements follow from the analysis of policy rules types, policy enforcement and composition mechanisms in Sections 2.3.3 - 2.3.5. Additionally, the following requirements were added: (i) the possibility to express constraints on attributes and unite them into complex condition expressions; (ii) the existence of specialised policy authoring tools which facilitate the policy authoring task.

According to the comparison presented in Table 2.1, only XACML satisfies all the requirements. This language, along with EPAL and KAoS, employs the outsourced enforcement model and, along with EPAL, has the required type of obligations. XACML and EPAL are quite similar in their aims and specification approaches. In spite of the fact that EPAL was designed with the focus on privacy, the functionality of EPAL v. 2.1 is a subset of XACML v. 2.0 and all significant EPAL concepts can be expressed using XACML [6]. Importantly, EPAL does not support hierarchical policy structures and policy composition, which facilitates policy specification especially when policies manage complex multidomain environments and originate from different sources.

Indeed, XACML is a flexible and extensible industry-strength policy language, which is widely used for a broad range of applications. It was adopted for the specification of policies in a number of open source environments, such as HERAS-AF and Axis2, and in some commercial platforms, like JBOSS, Axiomatics. Additionally, within the XACML community a number of open source PDPs and supporting tools were developed. It is argued (e.g., in [16]) that the disadvantage of XACML is its verboseness and it is hard to write and read policies in this language, due to the fact that it is an XML-based language. In order to mitigate this disadvantage, special authoring tools are being created that hide these complexities from the users [144][11]. Moreover, the usage of XML provides the possibilities to use XACML on different platforms and easily exchange with XACML policies.

2.4 Conclusion

In this chapter, a review of the e-Learning field was performed with the focus on the applicability of current technologies for the automatic CEP generation problem. ITSs and CG techniques, which provide an educational curriculum development functionality, were identified as the most

[10] The most recent version 3.0 was approved in January, 2013 [50].

[11] Some research-based initiatives even include development of a non-technical users notation for XACML policies [145].

relevant techniques within the e-Learning technologies stack. Strengths and limitations of their application to the CEP generation problem were analysed. It was found that the utilisation of hierarchical planning-based CG approach is the most advantageous for this task. On the other hand, limitations to the current approaches were identified that relate to the BP mechanisms support, the specification of CEP requirements, the utilisation of different terms and units representing the same notions, and others. These limitations should be resolved during the CEP generation solution design.

One of the main difficulties for the application of current techniques to the CEP generation problem is the need to specify and enforce heterogeneous educational regulations which are supported by different persons and which determine how a student mobility programme can be designed. It was shown that current CG as well as planning techniques do not provide sufficient mechanisms in order to satisfy the corresponding requirements. Policy-based management and policy specification languages were explored in this chapter as tools that support specification and enforcement of regulations developed by different parties independently. A comparison of existing policy languages was conducted and the XACML policy language was chosen for the specification of educational regulations and controlling a planning-based CEP development process.

Based on the outcomes of the presented review and analysis as well as on the conducted student mobility area analysis (see Chapter 3), a CEP development framework is proposed in Chapter 3. In this framework, we design the core process of CEP generation based on existing EPs and modules. The described restrictions of the current CG technologies are eliminated, and roles of different technologies, specifications and people for the CEP development are defined in this framework. The framework constitutes a basis for the design of a centralised solution that integrates HTN planning technologies and policy-based management for the generation of CEPs in environments with heterogeneous regulations (see Chapters 5 - 8).

	Required	Ponder	KAoS	EPAL	XACML
Policy enforcement	Outsourced or existence of application-independent PDP	Provisional, but for obligations application-independent PDP exists	Outsourced/ Provisional	Outsourced	Outsourced
Type of obligations	Generated during evaluation of authorisation rules	ECA obligations (reported to management agent)	ECA obligations (reported to autonomous agent or carried out on behalf of agent)	Generated during evaluation of authorisation rules	Generated during evaluation of authorisation rules
Modularity	Hierarchical policies or groups of policies	Several types of policy groups. Hierarchical groups are supported.	Non-hierarchical policy groups	None[12]	Hierarchical policy groups
Policy scoping	Scoping for policies and policy groups	Partially supported (using parametrised types of policies/policy groups)	None (scoping is a policy component)	None (scoping is a policy component)	Supported as a construct of higher level policy group
Policy composition	Support for different composition operations	Partially supported[13]	None[14]	None[15]	Supported
Attribute based decision	Supported	Supported	Supported	Supported	Supported
Conditions supported	Arbitrary expressions over attribute values	Arbitrary expressions over attribute values	Conjunction of atomic constraints on attribute values	Arbitrary expressions over attribute values	Arbitrary expressions over attribute values
Specialised policy authoring tool	Exists	Exists	Exists	Exists	Exists

Table 2.1: Policy languages comparison

[13] Non-hierarchical policies are supported only.
[14] For different policy groups distinct conflicts detection strategies can be used, i.e., application-specific meta-policies.
[15] Conflicts resolution is implemented using numeric priorities. Default authorisation decisions are specified per domains basis.
[16] Conflicts within policies are resolved using rules precedence.

Chapter 3

Framework for student mobility programmes development

Objectives:

- *Analyse the student mobility domain area.*
- *Specify requirements for a student mobility programmes development solution.*
- *Design a framework for student mobility programmes generation.*

3.1 Introduction

Nowadays, the student mobility field is a rapidly growing area [60]. Its development is facilitated by a number of major international initiatives, including the BP, the Erasmus and Erasmus Mundus Programmes, the Tuning project. In this chapter, we present a review of this area aimed at an elicitation of its main characteristics (see Section 3.2). First of all, an overview of different Combined Educational Programme (CEP) types that involve student mobility activities will be presented. A subsequent review includes an analysis of the main student mobility processes, a description of the main mechanisms proposed within the BP and an analysis of other characteristics and initiatives of the student mobility area. The results of this review will form the basis for the specification of the main requirements for a student mobility support solution providing the CEP generation functionality (see Section 3.3). Based on these requirements and the results of the current technologies review, presented in Chapter 2, the overall CEP generation framework is designed (see Section 3.4). This framework outlines a general CEP development process and determines roles and interrelations of different technologies, specifications and users in this process.

3.2 Student mobility domain area analysis

3.2.1 Student mobility processes analysis

In order to designate any educational pathway involving student mobility activities, in this study a 'Combined Educational Programme' notion is adopted. Extending the EP definition (see [131, 160]), a CEP is defined as an approved curriculum route incorporating student mobility activities that leads to one or several academic awards and is followed by a registered student. CEPs are classified [93][1] according to a number and types of qualifications, awarded at the end of the course:

Pure credits mobility degree. Only one qualification is awarded by one of the institutions offering the course. In the degree supplement, a part of the course taken at another education provider can be indicated.

Two majors or major and minor degrees (dual degrees). One degree with two majors (or one major and one minor), corresponding to the subject areas studied by the student, is awarded by one institution[2].

Joint degrees. One joint qualification is awarded by several institutions offering the course jointly.

Double (and multiple) degrees. Two (or more) individual qualifications are awarded by different institutions offering the course.

Student mobility processes within different types of CEPs are similar, but the design of these programmes and their goals are distinct. So in order to focus our attention on a specific case, in what follows we will concentrate on the pure credits mobility degree type of CEP. This CEP type reflects the nature of student mobility, and it commonly involves utilisation of existing modules and EPs without new educational content design.

The education of a student involving student mobility schemas is a complex process covering several stages and involving different participants (e.g., universities). We, therefore, adopt a business process modelling approach for the analysis of processes carried out when a student is studying according to a CEP. If we consider a credit mobility degree programme, there are two basic mobility schemas:

- Permanent transfer, when a student leaves one EP, moves to another EP and never returns to the previous EP (see Figure 3.1, B).
- Probation period, when a student moves to another university or EP, but after some period of education there (without graduation from this EP), he (or she) returns to the original EP, where he (or she) has studied before (see Figure 3.1, A)).

[1] An official universally accepted classification for these EPs is absent [122]. Different international HE experts use different terms and use different definitions for the same terms. We have adopted the classification presented in international reports [93] and [92] and extended it with 'credit mobility degree' CEP type from [91].

[2] This type of CEP usually involves only internal student mobility.

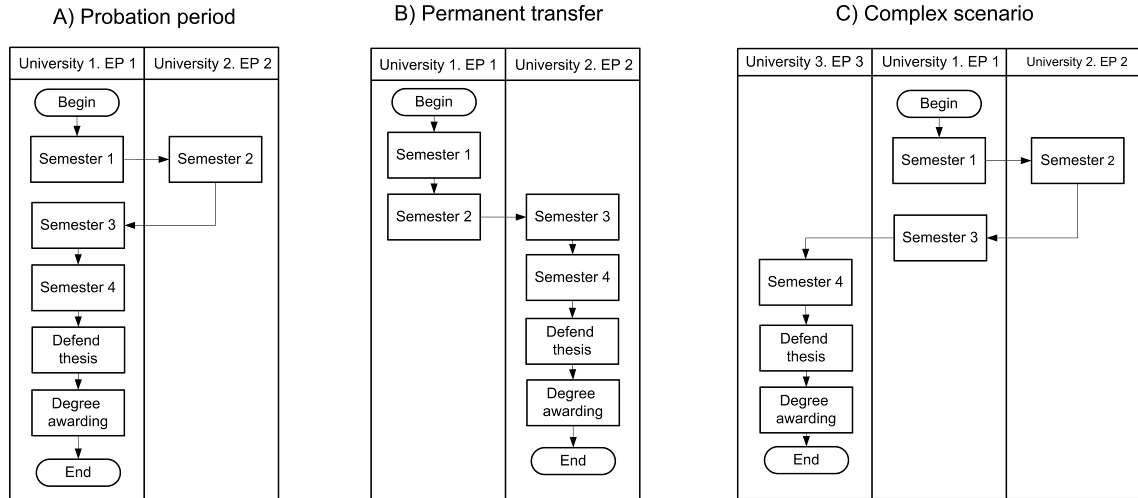

Figure 3.1: Student mobility schemas

Based on these basic schemas, more complex student mobility scenarios can be developed. These scenarios combine the basic scenarios to create more sophisticated CEPs. Such CEPs can include more than two universities and EPs [151]. An example of such a student mobility scenario is presented in Figure 3.1, C. First, the student starts the education at EP_1, but after the first semester a probation period in $University_2$ is taken. After the second semester, when studying at $University_2$, he (or she) returns to his (or her) 'home EP', EP_1, in the third semester. Finally, after the third semester, the student moves permanently to EP_3 in $University_3$, where a degree is granted for him (or her).

The student mobility scenarios have the following important characteristics:

- In different domains (e.g., countries, universities, faculties), there are different legal and normative requirements that affect how educational processes should be carried out. Therefore, the same high-level educational scenarios are implemented using different lower-level procedures within different domains. For example, when a student transfers from one university to another with the aim to receive a degree from the destination university, different procedures should be carried out based on the regulations of the university or faculty (e.g., modules studied in another university could be eligible or not for the recognition; for the recognition, the student could have to pass an interview or exam, or modules can be recognised without additional assessment).
- Results of the student mobility activities, as well as other educational activities in the educational environment(see [169]), are not known in advance. The execution of the designed EPs is always subject to the student's progress.
- When CEPs involving credit student mobility schemas are created, usually existing learning objects (e.g., terms of EPs and modules) are used as a basis for the CEP development.

This ensures a reuse of established university programmes, limits the expenses for the CEP development and eliminates the disruptions of the established university EPs. When existing EPs are utilised for the CEP development, they can be modified: some modules can be added or removed, or some optional modules can be done core modules for the student.

On the other hand, if learning objects (e.g., modules) are taken for the CEP development from different universities, they are not related to each other. In order to determine relations and align EPs from different universities (possibly in different counties) and recognise previous periods of education, a set of international initiatives has been started in the area of HE. They are examined in the following section.

3.2.2 International initiatives in the area of higher education

The Bologna Process (BP) is a well-known initiative aimed at the creation of the European Higher Education Area (EHEA). The BP provides mechanisms that help to harmonise degrees and quality assurance standards and make them more compatible and comparable. The student mobility is a core element in the BP initiatives. Intensification of mobility processes is one of the measurable results of the BP and one of the main components of the EHEA development [55, 84]. In this section, the main mechanisms proposed within the BP are described.

Cycles. The BP prescribes that all qualifications in HE should be located within three main cycles[3], which were established by a set of BP initiatives (see [55, 56]). Each cycle corresponds to a specific academic demands, complexity of knowledge, depth of learning and degree of student autonomy. A successful completion of a lower cycle gives access to a higher cycle. Additionally, a shorter cycle within or linked to the first cycle is distinguished. Adoption of these cycles facilitates the fair recognition of qualifications in different countries.

In the **Qualifications Framework of the European Higher Education Area (EHEA QF)** [17], the BP cycles and their descriptors are established as an overarching European framework. Based on this overarching framework, National Qualification Frameworks (NQFs) should be developed by each participating state (for example, see [132, 138]). A NQF should define all levels of education and types of HE qualifications awarded at each level of the national HE system[4]. Relations between these levels (or qualifications) should be described in the NQF, in concrete, points of their integration and intersection should be stated. Moreover, the NQF should map national qualifications and levels to the EHEA QF and BP cycles. Such system formed by the NQFs and the overarching EHEA QF makes the qualifications provided in different countries more compatible and comparable with each other.

[3]Informally, they refer to undergraduate, graduate and doctoral qualifications, but the BP cycles are indicated using numbers: first, second, third.

[4]Levels in terms of NQFs have the same meaning as cycles at the EHEA level. A NQF can introduce qualifications that does not equal to BP cycles, e.g., a cycle can be sub-divided.

According to the BP, each EP should refer to a NQF level with corresponding complexity and depth of study. The NQF level in turn should be related to an EHEA QF cycle. Such interrelations can be used during the student mobility programmes design for the comparison of EPs from different educational systems. First of all, based on the information about the NQF level of the qualification that a student has, this qualification can be recognised in order to satisfy the admission requirements of the receiving university. Secondly, the information about relations of EPs from different countries received from the qualification frameworks can be utilised to choose EPs that can be used as a basis for the mobility programme development. It should be noted that these relations in many cases are more complex than direct correspondence and cannot be determined precisely, as the NQF levels of these EPs can span across several BP cycles or correspond only to a part of the cycle.

Another important mechanism is the **Learning Outcomes-based education**. This mechanism facilitates the development of the competence and student-centred approaches to education. “Learning outcomes are statements of what a learner is expected to know, understand and/or be able to demonstrate at the end of a period of learning” [2], p.4. Within the BP, it is required that all levels in NQF, all EPs and all their components (modules, work placements, etc.) should be described in terms of learning outcomes. Using learning outcomes, the curriculum is described in terms of results acquired by the students without direct reference to the teaching process itself. Learning outcomes should be verifiable. In order to award credits for a learning unit, students should pass assessments evaluating the corresponding learning outcomes. So learning outcomes provide a link between EP planning, learning process, assessment and expectations of different stakeholders [66]. Learning outcomes make different qualifications and their components more comparable and facilitate the process of credits recognition.

According to the BP, each learning object (e.g., EP, semester or a module) should be described in terms of learning outcomes that specify results achieved during the education. The recognition of credits awarded for a learning object should be learning outcomes-based [1]. Decisions about relations of different learning outcomes can be taken as a result of analysis conducted by several experts with the experience in the data domain concerned.

Credits are defined in [54] as “quantified means of expressing the volume of learning based on the workload students need in order to achieve the expected outcomes of a learning process at a specified level” (p. 35). Credits are specified in terms of notional learning time[5] (notional hours). Credits should be allocated to the EP and to its constituent learning object (years, semesters and modules). Credits that are allocated to a learning object refer to its level and to the corresponding set of learning outcomes.

[5] “The notional learning time is the number of hours which it is expected a student (at a particular level) will need, on average, to achieve the specified learning outcomes at that level” [177], p. 239.

The **European Credit Transfer and Accumulation System (ECTS)** [54], which was developed within the BP, defines the main principles and requirements of credit allocation and credit transfer. Credits, allocated to a learning object, are awarded to individual students after the successful completion of this learning object. Accumulation of a specific number of credits is a compulsory requirement to award a degree. One ECTS credit is equal to 25-30 notional hours and one academic year is always 60 ECTS credits. In NQFs and programme specifications, ECTS credits should be used. If other credit units are used, a conversion scheme to ECTS credits should be provided.

Moreover, credits achieved according to one EP can be transferred to another EP. An institution awarding a degree to a student can **recognise** credits that he (or she) gained during the education according to an EP of different institution and exempt the student from studying some part of the EP leading to this degree. In this case, the institution considers that the requirements for credits and learning outcomes of this part of the EP are satisfied by credits and corresponding learning outcomes that were recognised. So credits are always recognised in relation to some EP[6]. Credits can also be **pseudo-recognised**: they are acknowledged by the awarding body as credits with adequate quality, but they are not counted towards the student's degree. These credits can be listed in the student's diploma supplement, but they do not reduce the number of credits that the student should receive at the awarding institution.

Another EHEA initiative consists of creating **networks of partner institutions**. Universities are encouraged to sign partnership agreements with other institutions, facilitating student mobility processes between these institutions. These partnership models are different from joint degrees, as they provide a basis for more flexible and student-centred mobility processes [67, 54]. Information about all the partners of the university should be publicly available [159, 15], so students and other universities can utilise this information when planning student mobility activities.

3.2.3 Other characteristics of student mobility domain area

The core process of EP design is defined as "a creative and often an innovative activity" [161], Chapter B1, p. 2. Usually, there is no precise and fixed procedure for the CEP development (neither for credit mobility CEPs, nor for other types of CEPs). The development of an EP is carried out based on specific expectations and requirements, available resources and local established principles. Official guidelines and recommendations (e.g, [1, 58, 130, 59, 64, 63, 159]) aimed at the student mobility facilitation include some specific, but usually uncoordinated guidelines covering different aspects of the CEP development. They try to solve a broad range of problems that occur during the CEP design and execution within academic, organisational, personal, financial and marketing fields. The majority of these recommendations are not relevant for the design of

[6]Moreover, the same credits achieved by a student can exempt him (or her) from studying parts of EPs with different number of credits, according to different recognition cases.

a CEP generation solution. For example, among these guidelines there are recommendations how to motivate students to take part in mobility programmes [67], how the funding issues should be resolved [63], how universities can select their partners [159], etc. But in these recommendations and guidelines there are some issues that should be taken into account during the CEP generation solution design:

- The awarding institution is responsible for the quality of all individual modules and other parts of the programme that are considered when the award is granted, regardless of the institutions which were involved in the learning process. If the awarding institution delegates the right to conduct some part of the EP to its partner and even if the partner delegates it further, the awarding institution should guarantee the quality of the education [159, 64]. For example, it is the awarding institution who decides about the recognition of credits from other institutions. Moreover, awards granted by an institution should conform to the statutory regulations and requirements of the controlling organisations, so it is the responsibility of the awarding institution to ensure this [130, 159].
- Universities should rigidly specify procedures and principles for making decisions concerning non-standard educational tracks, for example, accreditation of prior learning, advanced standing, etc. [130]
- In different domains, there are different legal and normative requirements to educational processes and to structures of EPs and CEPs. These requirements are compound: they consist of different sets of requirements, covering different aspects of the educational process and different cases, they are specified at different levels of the educational system and originate from different organisations.
- In different domains, different terms can be used to designate the same or similar notions and different units can be used to measure the same characteristics. There are variations in qualification levels, units of workload (e.g., in a value of credit point), grading scales, subject areas classifications, etc. This problem is acknowledged as one of the obstacles for the student mobility area development [151, 177].

3.3 Requirements to Combined Educational Programme development solution

As it was stated in Chapter 5.1, the aim of a CEP development solution is the facilitation of the student mobility area development (by lifting information, academic and financial obstacles). The main task of a CEP development solution is defined as follows. This solution based on existing modules and EPs should generate possible CEPs that satisfy the following requirements:

- They should be developed for an individual student or a group of students with similar characteristics.

- They should satisfy requirements provided by the requester of the CEP (an educational organisation or a student).
- They should satisfy regulations specified by the corresponding education providers and educational authorities.

The following requirements to a CEP generation solution can be specified based on the undertaken analysis of the student mobility area. CEPs that should be developed by the solution are usually targeted at individual students when the development is initiated by a student or a group of students with specific profiles when it is initiated by a university. Hence, the CEP generation solution should take into account information about *student characteristics* (like previous education, country of origin, language skills, etc.) for the CEP development.

As opposed to current CG techniques[7], within *the requirements for CEPs*, requirements for officially approved qualifications are equally or even more important than the requirements for learning outcomes that the student will achieve. Another important aspect for the institutional education carried out within the HE environment is the physical location of students during the education. When a CEP involving student mobility activities is developed, this factor becomes even more significant, as students should move physically during the education.

So for the *specification of CEP requirements* within the CEP generation solution a *flexible mechanism* should be developed. It should provide the means to specify requirements to award(s), gained during the education, structure of the CEP and physical movements of the student during the education.

For the generation of CEPs based on the user requirements, the CEP generation solution should support *different mobility schemas*. It should be possible to combine basic mobility schemas to form more complex student mobility scenarios, which satisfy the input requirements. Additionally, when a CEP is developed, all applicable *educational regulations* should be taken into account. These regulations can control different aspects of the educational process, for example, limit the set of possible student mobility scenarios, limit possible universities, EPs, modules, where a student can study. They can determine the structure of the CEP, determine how specific procedures should be carried out during the education (e.g., how the student transfer, admission or progression procedures should be carried out). The CEP generation solution should not only guarantee that the resulting CEP satisfies all the applicable regulations. It should provide the means for the specification and support of these compound regulations by different persons independently.

The CEP generation solution should use existing EPs as the basis for the CEP development. In order to compare EPs and modules from different universities, *the BP initiatives* should be employed. For each EP and module the learning outcomes, credits and NQF level should be specified. In order to filter which EPs can be used for the CEP development, the EHEA system of

[7] See the corresponding review in Chapter 2

educational frameworks should be used. CEPs should be constructed only from EPs corresponding to the same or overlapping educational levels. In order to compare individual modules or parts of EPs and make recognition decisions, their learning outcomes and credits should be compared. According to the educational recommendations, these procedures and criteria for the decision taking should be decided and fixed by the education providers. Therefore, they can be represented as educational regulations and utilised in the CEP generation solution.

Educational regulations and learning objects (e.g., EPs, modules) are usually specified using locally adopted units and terms that can differ in different domains, even if they refer to the same or similar notion or characteristic. So the CEP generation solution should support *different terms and units* used in the domains, covered by the solution.

3.4 Combined educational programmes generation framework

For the generation of CEPs within the CEP generation solution, a novel framework was developed based on the requirements described in Section 3.3. This framework determines at an abstract level how CEPs should be developed. It defines which technologies and specification instruments are utilised in this process, how they contribute to the CEP development and how they are interrelated. Moreover, it defines roles of different groups of users in the CEP development process. This framework combines planning techniques with policy-based management and rule-based approaches, providing the means to exploit the advantages of all these technologies. Using these components, based on existing EPs and modules, this framework can generate CEPs that satisfy user requirements and current regulations. A general architecture of the framework for the CEP generation is presented in Figure 3.2. The framework is divided into three layers each of which is responsible for its specific aspect. Each layer contains several key components interacting with one another and with the components in other layers.

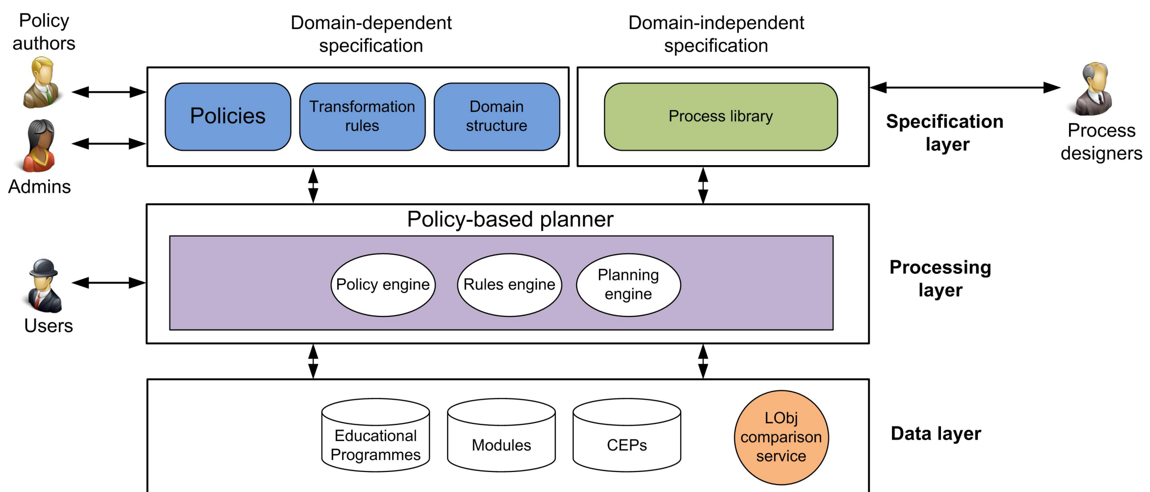

Figure 3.2: CEP generation framework

3.4.1 Specification layer

The first layer is a specification layer. It contains problem-specific information about the universe of discourse, that is, the HE environment and student mobility processes. In this layer, information about educational processes, mobility schemas, existing rules, routines and regulations is created, stored and maintained. This layer can be divided into two sub-layers. The domain-independent specification part represents common information about the universe of discourse, which is not related to any specific domain (e.g., a concrete university or country). This information is a description of common educational processes contained in a process library. Process descriptions are maintained by process designers, who should have corresponding competences in the problem area and process modelling. The domain-dependent specification part contains information specific to concrete domains. This information is represented as policies, transformation rules and the domain structure itself. It is created and updated by policy authors responsible for domain policies and system administrators responsible for specification of transformation rules and supporting the domain structure. Policy authors or administrators supporting one or several nested domains can belong to an organisation to which this domain corresponds (e.g., they can be members of the university or faculty).

The **process library** contains a set of process models, that is, descriptions of processes carried out when a student studies according to a CEP. These processes involve student mobility schemas and ordinary educational processes, carried out within one university. A process is defined in [74], p. 66 as "*a set of interrelated tasks that, together, transform inputs into outputs*". A process model is an abstract representation of a set of unified processes. Process models in the process library are represented as a set of (partially) ordered tasks. Process models can be specified at different levels of abstraction using compound tasks representing distinct sub-processes and primitive tasks that, on the contrary, do not have inner structure. Primitive tasks correspond to planning actions. They represent some actions that can be carried out in the environment leading to its transformation. Special method constructs specify the lower-level processes that compound tasks can be decomposed into and the conditions when this can be done. Using these methods, abstract process descriptions can be refined into concrete process specifications containing all the required details for their execution. Process model specification and enforcement mechanisms are built based on HTN planning and are presented in Chapter 5, while specific process models that were developed for the implementation of the CEP generation solution are described in Chapter 7.

The **domain structure** is a component that models a part of the HE environment that is supported by a CEP development system. This model has a multi-domain hierarchical organisation, meaning that it consists of nested domains forming a hierarchy. Domains in this model are universities, their internal units (i.e., faculties, schools), and higher level domains of the educational

environment (i.e., regions and countries). Mechanisms for the specification of domain structure are described in Chapter 7.

Policies specify educational regulations, routines and criteria for decision making that determine how the educational processes can be carried out and are taken into account during the CEPs development. Policies are always specified for a specific domain in the domain structure. Policies are compound, that is, they consist of several sub-policies composed using combining algorithms. Sub-policies within a domain policy can be used to represent regulations, managing distinct aspects of educational processes. For example, different policies can be specified for admission rules of the university, credits recognition rules, transfer routines for mobile student, degree awarding requirements, etc. Different policies, even within one domain, can be supported by different policy authors. Policies, specified by policy authors and saved into the policy repository, are enforced during the CEP development in a way that CEPs, produced by the CEP generation solution, should conform to all the policies applicable to them. Policies are used to limit the set of processes, specified within the process library, that can be executed during the CEP development. Additionally, using policies, these processes can be refined: new refined processes are produced that are specific to a current domain and according to the current regulations can be executed during the CEP development.

For the specification of policies in the CEP generation framework the XACML policy language was adopted. A formalised presentation of this language is provided in Chapter 4. Extensions of this language that enable its usage in our framework as well as principles of its utilisation are presented in Chapter 5.

Transformation rules are used to define relations between terms and units used in different domains (or classification systems) in order to designate the same or similar notions or measure the same characteristics of learning objects. Thus, the transformation rules are used to relate and compare learning objects specified within different domains (or using terms adopted in different classification systems). For example, using these rules, credit values and marks can be converted from the scale of one domain to another. Also they are used to map qualification levels between NQFs and EHEA QF levels and levels within NQFs of different countries. So in the CEP generation framework, these rules are utilised when a policy from one domain should be enforced for learning objects from another domain. Also they can used during the planning, when the conditions (within the process models) should be evaluated that operate with notions designated by different terms in different domains. For example, they can be used to select EPs from different domains that can be used for the CEP construction or to check that an EP award satisfies the user requirements (user requirements can also be specified in terms of different domains). Transformation rules and their invocation schema are described in Chapter 5. Examples of rules for the CEP development are given in Chapter 7.3.2.

3.4.2 Processing layer

The processing layer is the main operational layer of the framework grouping the core algorithms involved in solving problems posed by users. These algorithms utilise models stored at the specification layer and data objects and services from the data layer. Also in this layer, users interact with the system and utilise services provided by it. Users can be students or members of institutions who develop CEPs for students. A user provides requirements for a CEP and a profile for student(s) who will study according to this CEP. Based on this information, the solution, utilising learning objects stored at the data layer, should generate a set of CEPs that satisfy the user requirements and conform to all policies applicable to them. In more detail, this process is described in Section 3.4.4.

The main component of this layer and the core component of the framework, which was designed to carry out core solution building processes, is the **policy-based planner**. This component is based on three interacting engines. A planning engine carries out planning to solve the specified problem, utilising the process models from the process library. A policy engine evaluates and enforces policies, stored in the policy repository, to guarantee that they are not violated in the resulting plan. A rules engine carries out required transformations during the planning and policy evaluation utilising provided transformation rules. The core policy-based planner was implemented as a problem-independent engine, meaning that this component itself has not any knowledge about the problem area. All required knowledge is provided to it using the models at specification layer, including rules how data objects and services at the data layer can be used. During the further development of the planner, some extensions were introduced into it in order to bring gains in the planning performance, relying on the knowledge about the multi-domain hierarchical structure of the planning environment.

The basic version of the problem-independent policy-based planner is described in Chapter 5. The extension of this planner providing the means to evaluate policies at earlier stages of the planning is described in Chapter 6. This extension is used to improve the planning performance and utilised in the designed descending policy evaluation technique. This technique provides the means to evaluate policies for higher-level domains and optimises the process of learning objects selection during the CEP development, relying on the multi-domain structure of the planning environment (see Chapter 7).

3.4.3 Data layer

In order to construct a CEP, the policy-based planner requests information about existing EPs and modules from the data layer. This layer is used to store and operate with descriptions of Learning objects utilised or generated by the system. These Learning objects are CEPs, EPs and their components: semesters and modules. These objects and the main operations supported for

them are described in Chapter 7.

As was said before, for the comparison of credit values and qualification levels of Learning objects from different domains, transformation rules are used. For the comparison of Learning objects based on their leaning outcomes, a special **Learning objects (LObj) comparison service** should be used. This service should be based on an ontology-based similarity measure between learning outcomes[8]. Information about learning outcomes-based relations between Learning objects can be used for modules recognition and pre-requisites checks for mobile students, when pre-defined relations between the Learning objects are absent. As different criteria can be used in different domains to make these decisions, the Learning objects comparison service should support different measures between Learning objects, computed based on the values of the similarity measure for corresponding learning outcomes (see Chapter 7).

Thus, in the specification layer, problem-specific models are authored and supported. These models are utilised in the framework to represent and solve tasks within the concrete problem area, that is, the CEP development area. The processing layer groups the main computational algorithms, which are used to process the problem-specific models and apply them for problem-solving. Thereby, it links specifications and data objects, stored in the data layer. The data layer contains different data objects that can be used to build a solution in a specific problem statement.

3.4.4 An overview of the CEP generation process

As there is no adopted specific CEP development algorithm or approach in the educational area, a simulation approach was chosen for the CEP construction in our framework. At the centre of this approach, we place a process representation of the CEP that is constructed using the process models specified in the process library. This process model can represent information about activities carried out when a student is studying according to a CEP or activities relevant to this process. This information includes any relations of these activities and all its required parameters. This process can be modelled at the required level of detail at different stages of the CEP development.

The outcome of the CEP generation is a process representation of the CEP that models the educational process carried out according to this CEP (we will refer it *CEP process*). This CEP process model is constructed using process models using HTN planning, therefore, higher-level processes can be refined to lower level processes using methods that decompose compound tasks into sub-processes. Hence, as it is represented in Figure 3.3, the process of CEP construction can be viewed as a refinement of its process representation.

On the other hand, requirements provided as input data to the solution can represent the user's view on the future CEP from different perspectives and can be specified at different levels. So the

[8]This measure was developed as part of a contiguous research project and was described in [32]

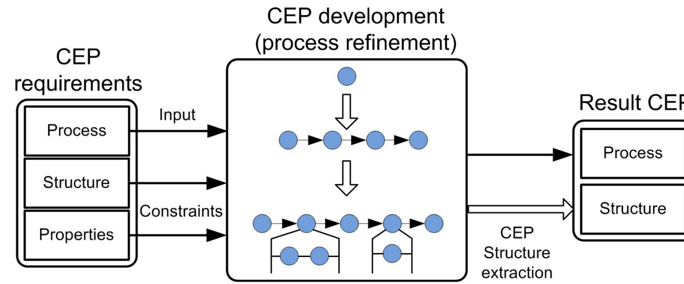

Figure 3.3: Overview of the CEP generation process

framework should support three types of input requirements: requirements for CEP process, CEP structure and CEP properties. The requirement for the CEP process is a high-level CEP process specification. If a user has not requirements for the CEP process, a single highest-level task can be used as an initial CEP process specification. The requirements for the CEP's structure are represented as an Initial track (ITr). ITr specifies in which domains the student will study and how he (or she) will transfer between them. In ITr, domains at different levels can be used, so it can specify requirements to a physical track of the student at different levels of abstraction. Finally, a user can request that the CEP should satisfy some constraints specified for its properties (e.g., on its award, its duration or its resulting learning outcomes), that is, the CEP property requirements.

The abstract specification of a CEP process, which was provided by a user, is used as a basis for the CEP generation. It is refined using process models from the process library in order to receive a fully specified CEP process. During this process, different mobility schemas represented as process models are tried. These process models are designed in a way that they can be combined to form arbitrary complex mobility scenarios. Tasks used within the CEP process specification use as parameters specific EPs, their semesters and modules designating that they should be studied by the student or other actions should be carried out with them. During the refinement process, the planner should try different EPs, semesters and modules in order to build a fully specified CEP that satisfies the provided requirements for structure and properties.

The CEP process simulates the educational process of the student according to a CEP. In order to guarantee that this process is feasible and can be executed in the HE environment, we should check that it conforms to the constraints specified in the process library and to all policies applicable for it. Policies can be specified by members of different universities or higher-level organisations and represent their requirements to the CEP process. Hence, enforcing them during the CEP generation, we ensure that the resulting CEP meets the expectations of institutions that will be involved in the educational process according to the CEP.

As a result of this refinement, we get a fine-grain CEP process representing all the information about the CEP. A more concise and common representation of the CEP, that is, its structure, can

be extracted from the CEP process model at the end of the refinement. It represents modules of this CEP grouped into semesters in a concise table form.

3.5 Conclusion

First of all, in this chapter the student mobility domain area was reviewed, different types of student mobility programmes, international initiatives for the student mobility area development and their mechanisms were considered. Important characteristics and peculiarities of this domain area were distinguished. These characteristics were used as the basis for the requirements specification for a solution providing the CEP generation functionality. Subsequently, these requirements are taken into account during the development of such solution, in concrete, in this chapter they were utilised for the CEP generation framework design. The main contribution of this chapter is the design of the framework for the automated development of new CEPs using existing EPs and modules in an environment with heterogeneous regulations. As was shown in Chapter 2, the CEP generation problem was not considered before within the e-Learning field, so the designed framework constitutes a new approach for the support of student mobility processes using computer technologies. Although a similar problem of non-mobile curricula development is being solved using CG techniques, their application within the new area, the student mobility field, is limited by a number of factors. These limitations include a lack of support for heterogeneous educational regulations developed by different persons, for different terms and units designating the same notions, a need for a student mobility scenarios development mechanism, for a flexible mechanism for the specification of CEP requirements, and others. Within the CEP generation framework design, mechanisms for the elimination of these restrictions are provided. The framework combines the hierarchical planning technologies with the policy-based management and rule-based approaches, providing the means to exploit the advantages of all these technologies.

In the subsequent chapters, components of this framework are described in more detail. Chapter 4 presents the XACML policy language, which will be adopted for the specification of educational regulations in the framework. Chapters 5 and 6 present the design of the problem-independent policy-based planner, which will be used in the framework as its core engine. Chapter 7 describes the problem-dependent specifications that will be utilised in the framework and will enable the policy-based planner to solve the CEP generation planning problems.

Chapter 4

XACML policy specification language formalisation

Objectives:

- *Present the XACML policy specification language.*
- *Analyse the XACML policy language and the corresponding processes of policy evaluation and enforcement. Construct a formalisation for the XACML policy language that can be used for its extension.*

4.1 Introduction

In this chapter, we consider the eXtensible Access Control Markup Language (XACML) policy language that was chosen in Chapter 2 as a tool for the extension of automated planning techniques in order to carry out planning in environments with heterogeneous regulations supported by different persons independently. Correspondingly, in the CEP generation framework, which was proposed in Chapter 3, this policy language will be used for the specification of educational regulations which govern the educational processes, including the student mobility processes. The XACML policy language is an XML-based authorisation policy specification language supporting obligations generated during authorisation checks. The XACML policy language was approved as an OASIS standard policy language. Its prominent characteristics include extensibility, the support for different combining algorithms, using which policies and policy groups being specified independently can be composed to form higher-level policies, and the support for attribute-based decisions with the possibility to specify arbitrary complex conditions using a wide range of functions.

When describing the XACML policy language, we also introduce a formalisation for XACML policies and their evaluation algorithm. In the official specifications of the XACML policy language ([153, 50]), the semantics of the language constructs is introduced using natural language. The formalisation of XACML proposed within this study is needed when the policy-based planner, where XACML is used for the specification of policies, is extended and the postponed policy

enforcement mechanism is introduced (see Chapter 6). For the realisation of this mechanism, the XACML policy language and its evaluation algorithm should to be extended with the possibility to evaluate policies in situations when not all relevant information is available. During this extension, the formalisation is used to introduce the extended policy evaluation algorithm and analyse its properties (including its interrelations with the original version). Additionally, due to the verbose XML-based syntax of XACML policies, this formalisation is utilised in this and the subsequent chapters in order to represent policies in a concise form.

According to its main goal, the formalisation of XACML should concentrate on the representation of the XACML evaluation algorithm. It should be possible to modify the evaluation algorithm and to analyse its properties. Additionally, for the completeness of the analysis, the formalisation should cover all levels of the XACML policy structure. A number of formalisations for the XACML policy language were already proposed. In contrast with our case, the aim of these formalisations is usually to provide the tools that can analyse the properties of policies themselves. The common approach is to represent the policies to be analysed using an established formal representation and employ its existing reasoning mechanisms in order to analyse the properties of policies, including inference of policy decisions for specific queries. The formal representations include description logic [142], binary-decision diagrams [62], defeasible logic [94] and other approaches. Current formalisation approaches have several different drawbacks relatively to our goal. First of all, as they are aimed at the policy analysis, that is, not the analysis of the policy evaluation algorithm, the process of policies and policy constructs transformation to formal objects is usually not introduced as a formal mapping and is not based on the syntax structure of policies, so possible policy structures are implied. Other limitations are an absence of formal definitions for policy constructs themselves, regardless of a specific policy that is formalised at the moment (this makes impossible to analyse the policy evaluation algorithm itself, eliminating specific policies, e.g., in [62]), an elimination of important aspects of the policy evaluation, which are not crucial for the business meaning of the policies (e.g., exceptions that can be raised during the evaluation and which are represented using special Indeterminate decision in XACML in [94, 62]), formalisation of just some part of XACML policy structure (e.g., only combining algorithms are formalised in [121, 99], rule conditions are simplified to Boolean expressions in [62, 94]). Our formalisation is based on the context-free grammar for abstract syntax that was introduced for XACML. This grammar defines possible structures for XACML policies. When the formalisation is introduced, all its possible productions representing different XACML syntax constructs are mapped to objects representing them formally. These objects, effectively, formalise the process of these constructs' evaluation. The formalisation is defined in a compositional manner, so the meaning of each construct is defined in terms of the meaning of its components. Thus, the properties of the whole XACML evaluation algorithm can be analysed using the structural induction: it is required to analyse each individ-

ual construct and infer properties of their composition relying on the structure defined using the abstract syntax grammar.

The chapter is organised as follows. Section 4.2 contains an overview of the XACML policy language and the corresponding policy enforcement process. In Sections 4.3 and 4.4, the policy evaluation process and the introduced abstract syntax grammar for XACML are described. In Section 4.5, we describe different XACML constructs and provide their formal definitions. Finally, in Section 4.6 the obligations processing routine is described.

4.2 XACML policy language overview

XACML uses the outsourced enforcement mechanism, based on the PEP/PDP architecture (see Chapter 2). The XACML specification includes the normative requirements for application-independent PDP, which carries out the functions of the policy evaluation engine, and possible variants of the PEP behaviour, which is usually part of the application. PEP controls actions carried out in its area of responsibility. When policies should be evaluated, it queries the PDP creating and sending a policy decision request. The PDP evaluates this request, analysing policies stored in its repository, and infers a decision that contains an authorisation decision and, optionally, a set of obligations. In turn, the PEP enforces the decision: it permits or blocks the action and executes the obligations returned.

Correspondingly, the policy evaluation in XACML is separated into two phases: policy decision request (policy request) generation and policy request evaluation. For evaluation in the PDP, the policy request should contain a part of information about the action, which will be used by the PDP to infer a policy decision. The structure of the policy request in XACML is aligned with the structure of the authorisation policies and contains four sections: subject, action, resource and environment, implying that a subject carries out an action on a resource in the presence of some environmental parameters. Each section contains a set of attributes that describe the corresponding entity. As XACML is a typed language, all attributes in the policy request are stored along with their types. Information about the action that cannot be represented using attributes or it is difficult to do, such as the structure of the resource or its content, can be placed into the request as a free-structure XML document. So the policy request can contain arbitrary details about the action, including its subject, resource and environment, where it is being executed, that can be taken into account during the policy decision inference. Correspondingly, XACML policies specify conditions over information contained in the policy request that should be satisfied in order to infer certain policy decisions. Obviously, in this schema, it is possible to provide to the PDP through the policy request only a part of the information that will be actually used during the policy conditions checks. Moreover, limiting the policy request size is advantageous as this reduces the communication expenses, reduces expenses for the policy request formation (some information

could be explicitly retrieved from external sources) and reduces the time needed to retrieve the required information from the policy request during the policy evaluation. On the other hand, when a policy request is generated, there is no insight regarding what information can be required during the policy evaluation and, correspondingly, what information should be contained in the policy request. When a required attribute is missing during the evaluation, an indeterminate decision is produced along with which a description of the missing attribute can be provided. If the missing information can be provided, it is added into the request and the request should be re-evaluated[1]. As part of the policy-based planner design, this drawback was eliminated and the possibility to generate XACML policy requests considering the information that can be required during the policy evaluation was provided (see Section 5.6).

There are several authorisation decisions in XACML. In addition to Permit and Deny decisions, an Indeterminate or Not Applicable decision can be returned. Indeterminate is returned when a deterministic decision cannot be inferred because of some error (incorrect types of values in the policy request, errors in policy specifications, etc.) Not Applicable means that none of the policies can infer a decision for this policy request. When an Indeterminate or Not applicable decision occurs, PEP's behaviour is determined by the type of PEP, for example, Permit-biased or Deny-biased (e.g., Deny-biased PEP permits the action only if Permit was returned and it can discharge all returned obligations and blocks the action otherwise).

Obligations in XACML are represented as a set of actions produced along with the authorisation decision. These actions should be executed by a PEP in conjunction with the enforcement of this decision. In XACML, there is no means to specify a specific routine for how these actions should be executed, for example, a possible order of their execution or a position of their execution relatively to the action that was requested (before, during or after). This could be required, for example, when several obligations are returned that should be executed in a specific order, when some obligations should be executed after the requested action, for example, in order to save results of the execution in a log. When XACML is used in the policy-based planner for the specification of policies enforced during the planning, this is even more important as the plan which is being developed should contain all actions that should be carried out to achieve a goal and specify all ordering between these actions significant for its correct execution. During the development of the policy-based planner, the XACML policy language was extended and this drawback was resolved (these extensions are described in Chapter 5).

In this study we utilise the XACML policy language version 2.0, as at that time when the study was done it was the major version approved as an OASIS standard for which an open source policy

[1] Alternatively, for some missing attributes the PDP can support the function of its request during the evaluation. This function cannot be provided for the information stored as a free-structure XML document within the request (as it is requested using an XPath expression). Moreover, these requests should be done only for distinct attributes and lead to extra expenses.

evaluation engine was available. The XACML version 3.0, which is the most recent version at the time of writing, was a developing version at that time: it was not approved yet as a standard[2] and a policy evaluation engine was not available for it.

4.3 Policy evaluation schema

XACML policies are specified in a modular and compositional manner, namely, individual policies can be specified independently and composed into more complex policies. An example of XACML policy set is presented graphically in Figure 4.1[3]. The major components of the language are rules, policies and policy sets. A rule contains an effect part that determines an authorisation decision, returned when this rule triggers, and a condition part (i.e., a condition expression over information in the policy request), that determines when this rule can trigger. Policies are used to group related rules. In turn, policy sets are used to group related policies. Each rule, policy and policy set also contains a target element. Target is a condition with a restricted structure that determines when this construct is applicable to the policy request. Using the target mechanism, the scope of a specific policy and policy set is restricted. All components nested into a policy or policy set are evaluated only when the condition in its target is satisfied. For example, in Figure 4.1, a condition in $Target_0$ can specify that this policy is used for the specification of educational regulations within some university. Correspondingly, conditions in $Target_1$ and $Target_2$ can be used to specify that $Policy_1$ contains rules for the faculty of Humanities and $Policy_2$ - for the faculty of Computer Science. Other policies can also be included into this policy set, for example, the overall university policies. Each policy (or policy set) uses some rule (or policy) combining algorithm that determines the routine for processing the evaluation results for components nested into the policy (or policy set). This routine resolves conflicts between them and determines a resulting decision. In order to designate which combing algorithm should be used for a policy (or policy set) an identifier of the algorithm is provided within its structure (designated using greyed rectangles with rounded corners in Figure 4.1). Using these mechanisms, the specification of different policies and policy sets can be delegated to different authors who can specify the policies independently. Additionally, using different policies, regulations that manage different aspects of the system behaviour can be easily specified. During the policy evaluation, all components nested into a policy and policy set are considered only when its target condition is satisfied. Moreover, different policies can be specified in different XML files using policy referencing. Each XACML policy has its identifier and, instead of specification of nested policies in the same file, policy identifiers can be used to refer to policies

[2]The XACML policy language version 3.0 [50] was approved as an OASIS standard on 22 January 2013. The new version left the core functionality of XACML unchanged, but introduced the means for the policy author to specify policies in a more flexible way, relaxing some previously introduced restrictions in the policy specification schema. Major new functionality was introduced as separate or updated profiles (i.e., the administration and delegation profile and the multiple decision profile).

[3]As XACML policies are specified as XML documents, for the aim of their concise and demonstrative representation a graphical representation is used.

specified in separate files.

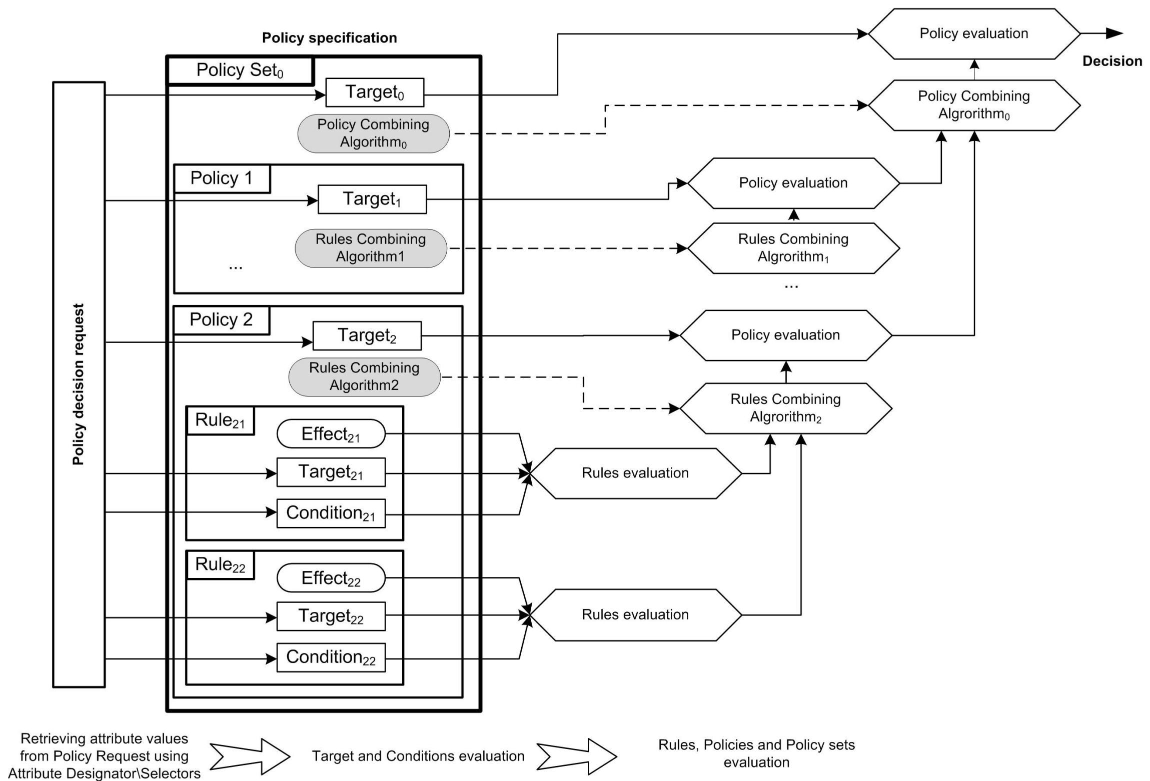

Figure 4.1: An overall schema of policy evaluation

The overall process of XACML policy evaluation was analysed and its structure is represented schematically at Figure 4.1. In the lower level of the evaluation (represented at the left side of the figure), values of attributes are retrieved from the policy request using two types of components: attribute designator and selector. The attribute designators and selectors are used within target and condition expressions and supply values for their evaluation. The second phase of the policy evaluation is the evaluation of target and conditions expressions of policies, policy sets and rules. The results of their evaluation are boolean values. These values are processed according to the semantics of rules, policies and policy sets. In the XACML standard [153], the semantics of each policy construct is described and results that should be produced during their evaluation are specified. In Figure 4.1, this phase is represented as a composition of elements representing the evaluation of rules, policies and policy sets. The structure of this composition is determined by the policy structure. As different policy and rule combining algorithms are supported, combining algorithm that should be used during evaluation of a concrete policy and policy set is determined based on its identifier specified as part of the policy or policy set (designated using dotted lines). Formal definitions for these components will be given in Section 4.5.

4.4 Abstract syntax of XACML policies

The first part of a grammar for the abstract syntax of an XACML policy is presented in Figure 4.2. This grammar and corresponding abstract syntax was designed based on the description of the XACML policy language in its specification [153]. It provides the means to distinguish distinct syntactic constructs of the XACML policy language and determine correct ways of their composition into XACML policies. The grammar was designed in the focus on the representation of valid composition of the XACML constructs. For this purpose, obvious possibilities for the abstraction of similar constructs using the same non-terminals (e.g., distinct non-terminals were used for different combining algorithms), what would reduce the number of non-terminals and make the grammar itself more compact, were omitted. The rest of the grammar is presented in Section 4.5.5. For the demonstration purposes, only a subset of the XACML policy language (representing its core functionality) was considered, viz., only two policy and rule combining algorithms (ordered permit- and deny-overrides), no combining parameters are supported and no optional construct in a policy can be omitted (e.g., its target). A mechanism for the obligation support is described separately in Section 4.6.

```
(1) PolicySet ::= CombPO Target PolicyCombPO | CombDO Target PolicyCombDO
(2) PolicyCombPO ::= Policy | Policy PolicyCombPO | PolicySet | PolicySet
PolicyCombPO
(3) PolicyCombDO ::= Policy | Policy PolicyCombDO | PolicySet | PolicySet
PolicyCombDO
(4) Policy ::= CombPO Target RuleCombPO | CombDO Target RuleCombDO
(5) RuleCombPO ::= Rule | Rule RuleCombPO
(6) RuleCombDO ::= Rule | Rule RuleCombDO
(7) Rule ::= Effect Target BoolVal
(8) Effect ::= Permit | Deny
```

Figure 4.2: Abstract syntax for XACML policies

An example of the Abstract Syntax Tree (AST) produced using the developed grammar for the policy set example in Figure 4.1 is presented in Figure 4.3 considering that $PolicySet_0$ is using the permit-overrides policy combining algorithm, $Policy_1$ - deny-overrides rule combining algorithm and $Policy_2$ - permit-overrides. As follows from the defined grammar, policy sets can contain other policy sets and policies, while policies can contain only rules that produce effects specified in their effect part (either Permit or Deny). One policy or policy set can use only a single combining algorithm, which is used to process all constituent rules or policies. The rule's condition is represented in a grammar as a $BoolVal$ non-terminal that represents any possible expression that produces a boolean value (true or false).

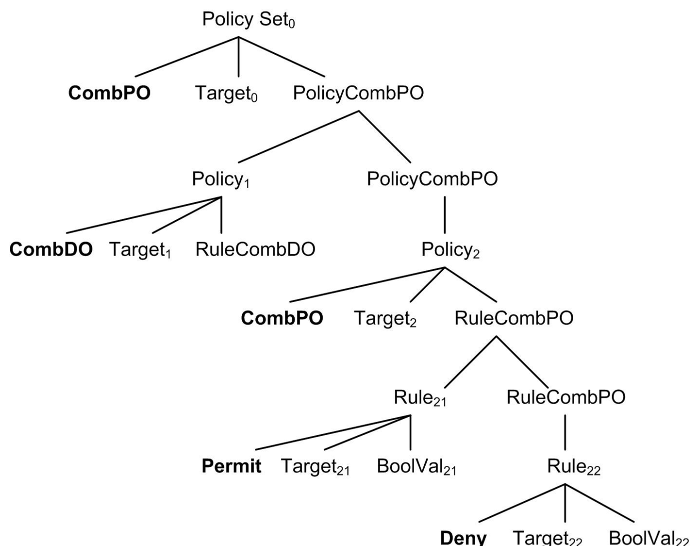

Figure 4.3: Example of AST

4.5 A formal model of policy evaluation

In this section, the introduced formalisation of the XACML policy language is described. Within this formalisation, for policy constructs specified using the abstract syntax grammar in Section 4.4 their formal definitions are given. These definitions specify how this policy construct should be processed during the policy evaluation, that is, how a result of its evaluation is derived based on the evaluation results of its constituent components. In order to device a formal definition for the policy or policy set evaluation routine, it is required to define sets of all possible values used as its input and output. According to the XACML specification, we will define a set of all possible outcomes (decisions) of policy and policy set evaluation as the set $M_1 = \{P, D, Ind, N/A\}$, where P and D are Permit and Deny decisions, Ind and N/A are Indeterminate and Not Applicable decisions. When a policy is evaluated, the evaluation results for its constituent rules are used as input information. In order to represent the rule evaluation results, we have extended the set M_1 into the set M_2: $M_2 = \{P, D, PInd, DInd, N/A\}$, where $PInd$ (or $DInd$) denotes that a rule decision was Indeterminate and in its $Effect$ part Permit (or Deny) is specified. This is required, because, as opposed to policies, when rules evaluation results are combined using some rule combining algorithm during the policy evaluation, it is important to know which decision, Permit or Deny, is specified in its $Effect$ part (i.e., it is a decision that this rule could have returned).

Among other data types used in XACML, truth values have a special role in the policy evaluation as they complete the chain of policy conditions evaluation. That is, target and condition constructs produce boolean decisions which are used at the higher level of some rule or policy

evaluation, for example, to decide if a rule should fire, or if a policy is applicable to the policy request. A set of truth values contains three elements in XACML: $TRVal = \{True, False, Ind\}$. The indeterminate value Ind is used to designate that during evaluation of an expression an error occurred and a concrete value cannot be decided. The set of values with other values data types, which are supported by XACML, is represented as a set VAL. In XACML, errors that occur during the evaluation of expressions should produce an Indeterminate value. So each type of values has a special element Ind, representing that an error occurred during evaluation of an expression with specific data type, for example, for integers it is Ind^{Int}.

The evaluation of policies is carried out over a policy request provided. A policy request is defined as a finite list of attribute values $attVal$ and a tree-structure $tree$, which represents an XML document providing additional information for the policy evaluation: $req = \langle\langle attVal_1, \ldots,$ $attVal_n\rangle, tree\rangle$. A set of possible requests is designated as REQ. Each attribute value $attVal$ is an element of set $Identifier \times Category \times (VAL \cup TRVal)$, where $Identifier$ is the set of all possible attribute identifiers, $Category = \{Subject, Resource, Action, Environment\}$ designates which entity this attribute characterises. The value of the attribute is within a union of sets VAL and $TRVal$.

In the following sections, formal definitions for procedures carried out when different policy expressions are evaluated over a policy request are given. These definitions are introduced as semantic functions mapping syntactic policy constructs, specified using the abstract syntax grammar, to values which represent outcomes of their evaluation. Semantic functions are uniformly named as $evaluate$ with a superscript designating a syntactic construct that it applies to (e.g., $evaluate^{PS}$, $evaluate^{T}$, $evaluate^{Rule}$). Correspondingly, policies, policy sets and policy combining constructs are mapped to values in the decision set M_1, rules and rule combining constructs - to values in the decision set M_2 and targets and rules conditions - to values in the truth values set $TRVal$. Semantic functions will be specified using semantic equations defining how semantic functions behave on different patterns of the syntactic expressions, where each pattern usually corresponds to one production of the abstract syntax grammar. XACML syntactic expressions within the semantic equations will be represented using strings of terminals and non-terminals of the abstract syntax grammar. In order to distinguish them from the formal definitions, syntactic expressions will be enclosed within emphatic brackets (e.g., ⟦**`CombPO`** `Target PolicyCombPO`⟧). Sets of all possible syntactic objects represented in the grammar by one non-terminal will be represented using italicised designation of this non-terminals (e.g., $PolicySet$).

4.5.1 Policy set evaluation

A policy set evaluation procedure is defined using the function $evaluate^{PS} : PolicySet \times REQ \rightarrow M_1$. This function maps syntactic constructs for policy set and a policy request to a decision

Table 4.1: Table of values for Policy evaluation function P^e

Target \ Combining decision	P	Ind	D	N/A
$False$	N/A	N/A	N/A	N/A
Ind	Ind	Ind	Ind	Ind
$True$	P	Ind	D	N/A

from the set M_1 that will be produced during the evaluation of this policy set over the given request. Semantic equation is given for this function in Formula 4.1, where the first production rule at row 1 in Figure 4.2 is considered[4]. As it can be seen, the target is evaluated first ($evaluate^T(\llbracket\texttt{Target}\rrbracket, req)$). It determines if this policy set is applicable to the request. If the $False$ or Ind value is received, the policy set is not evaluated further. If the target is satisfied, the evaluation result for the policy set is determined by combining the constituent policy and policy sets. If we eliminate information about the lazy evaluation[5] of policy set expressions, the $evaluate^{PS}$ function where permit-overrides (PO) combining algorithm is used can be represented as $evaluate^{PS}(\llbracket\texttt{CombPO Target PolicyCombPO}\rrbracket, req) = P^e(evaluate^T(\llbracket\texttt{Target}\rrbracket, req),$ $evaluate^{CombPol^{PO}}(\llbracket\texttt{PolicyCombPO}\rrbracket, req)$ using the auxiliary function $P^e : TRVal \times M_1 \to M_1$, which is defined on the decision and truth value sets. The truth table for this function is represented in Table 4.1. When a production rule with the deny-overrides (DO) is used, the same functions are used to formally define the procedure for its evaluation with an exception that $evaluate^{CombPol^{DO}}$ is used instead of $evaluate^{CombPol^{PO}}$. In what follows, $evaluate$ functions are defined directly using auxiliary functions on the decision and truth values sets with the assumption that expressions are evaluated from left to right in a lazy manner.

$$\begin{aligned} evaluate^{PS}(\llbracket\texttt{CombPO Target PolicyCombPO}\rrbracket, req) = case\ evaluate^T(\llbracket\texttt{Target}\rrbracket, req)\ of \\ False : N/A \\ True : evaluate^{CombPol^{PO}}(\llbracket\texttt{PolicyCombPO}\rrbracket, req) \\ Ind : Ind \end{aligned} \tag{4.1}$$

4.5.2 Policy and policy set combining algebras

Policies and policy sets produce decisions from the set M_1 during the evaluation. When these policies and policy sets are composed into a policy set using a combining algorithm, these decisions are combined and a resulting decision from the set M_1 is produced. In this formalisation, this combining process was separated into pairwise policy decision combinations. These pairwise

[4]In this rule the permit-overrides policy combining algorithm is utilised.

[5]An evaluation strategy is lazy when it delays evaluation of an expression until it is needed. In our case, if a value of a function is uniquely determined by already evaluated sub-expressions, other its sub-expressions should not be evaluated.

Table 4.2: Tables of values for Permit-overrides and Deny-overrides policy combining operations $\bullet_p^{PO}$ and $\bullet_p^{DO}$

$\bullet_p$	P	Ind	D	N/A
P	P	P	P	P
Ind	P	Ind	Ind	Ind
D	P	Ind	D	D
N/A	P	Ind	D	N/A

$\bullet_d$	D	Ind	P	N/A
D	D	D	D	D
Ind	D	Ind	Ind	Ind
P	D	Ind	P	P
N/A	D	Ind	P	N/A

combinations are formalised using the semantic function $evaluate^{CombPol^{PO}}$ or $evaluate^{CombPol^{DO}}$ (see Formula 4.2). These functions are applicable to syntactic constructs for permit-overrides and deny-overrides combining algorithms correspondingly (see rules at rows 2 and 3 in Figure 4.2). In Formula 4.2, the semantic equation for the case when a policy is combined using the permit-overrides combining algorithm is presented. This combining algorithm is formalised using the auxiliary function $\bullet_p^{PO}$. If a policy set is used, the function $evaluate^{P}$ is substituted by the function $evaluate^{PS}$. If the deny-overrides combining algorithm is used, the function $\bullet_p^{DO}$ should be used. When the `PolicyCombPO` abstract grammar non-terminal is substituted by a single policy (or policy set) the function $evaluate^{CombPol^{PO}}$ is defined according to the following semantic equation: $evaluate^{CombPol^{PO}}([\![\texttt{Policy}]\!], req) = evaluate^{P}([\![\texttt{Policy}]\!], req)$.

$$evaluate^{CombPol^{PO}}([\![\texttt{Policy PolicyCombPO}]\!], req) = \\ evaluate^{P}([\![\texttt{Policy}]\!], req) \;\; \bullet_p^{PO} \;\; evaluate^{CombPol^{PO}}([\![\texttt{PolicyCombPO}]\!], req) \tag{4.2}$$

When the permit-overrides combining is used, if a policy has returned Permit this decision overpowers all other policy decisions and is returned as a decision of the policy set (see Table 4.2). The Indeterminate decision has a priority over all decisions with the exception for Permit, because it is implied that the Indeterminate decision could be replaced with Permit if the error had not occurred. The deny-overrides policy combining is defined according to the same principle, but Permit and Deny are swapped.

As the domains and co-domains of the functions that model the policy combining are equal, these functions are operations. So it is possible to formalise policy combining algorithms as an algebra A_p with two binary operations $\bullet_p^{PO}$ and $\bullet_p^{DO}$.

$$\mathcal{A}_p = \langle \mathbf{M_1}, \{\bullet_p^{PO}, \bullet_p^{DO}, N/A, P, D\}\rangle \tag{4.3}$$

where

$\bullet_p^{PO} : M_1 \times M_1 \rightarrow M_1$ - operation for permit-overrides combining.

$\bullet_p^{DO} : M_1 \times M_1 \rightarrow M_1$ - operation for deny-overrides combining.

N/A, P and D - three special nullary operations, representing designated elements.

The tables of values for the binary operations $\bullet_p^{PO}$ and $\bullet_p^{DO}$ (see Table 4.2) are symmetric and values at their diagonals are equal to the arguments. These values show that these operations in a complex combining expression, like $a \bullet_p b \bullet_p c$, can be evaluated in any order: a 'maximum' operand is always returned. So the following properties hold for the binary operations $\bullet_p^{PO}$ and $\bullet_p^{DO}$ (both are designated as $\bullet_p$):

- Commutative: $a \bullet_p b = b \bullet_p a$.
- Idempotent: $a \bullet_p a = a$.
- Associative: $(a \bullet_p b) \bullet_p c = a \bullet_p (b \bullet_p c)$.

So the magmas $\mathcal{L}_p^{PO} = \langle \mathbf{M_1}, \bullet_p^{PO} \rangle$ and $\mathcal{L}_p^{DO} = \langle \mathbf{M_2}, \bullet_p^{DO} \rangle$ are semigroups and semilattices. As they are semilattices, natural orders for them can be defined as $a \leqslant^{PO} b \Leftrightarrow a \bullet_p^{PO} b = b$ and $a \leqslant^{DO} b \Leftrightarrow a \bullet_p^{DO} b = a$. A natural order for Permit-overrides semilattice is represented in Formula 4.4 and for Deny-overrides semilattice - in Formula 4.5.

$$N/A \leqslant^{PO} D \leqslant^{PO} Ind \leqslant^{PO} P \tag{4.4}$$

$$N/A \geqslant^{DO} P \geqslant^{DO} Ind \geqslant^{DO} D \tag{4.5}$$

As these orders are not dual, the algebra $\mathcal{L}_p = \langle \mathbf{M_1}, \bullet_p^{PO}, \bullet_p^{DO} \rangle$ is not a lattice and absorption laws $a \bullet_p^{PO} (a \bullet_p^{DO} b) = a$ and $a \bullet_p^{DO} (a \bullet_p^{PO} b) = a$ do not hold for $\bullet_p^{PO}$ and $\bullet_p^{DO}$[6].

The nullary operations N/A, P and D in the algebra $\mathcal{A}_p$ were included in it to designate special elements. The decision N/A is a universal identity element for the operations $\bullet_p^{DO}$ and $\bullet_p^{PO}$:

$$\begin{gathered} \forall a \in \mathbf{M_1} \ (a \bullet_p^{PO} N/A = N/A \bullet_p^{PO} a = a) \\ \forall a \in \mathbf{M_1} \ (a \bullet_p^{DO} N/A = N/A \bullet_p^{DO} a = a) \end{gathered} \tag{4.6}$$

The decisions P and D are adsorbing elements for the operations $\bullet_p^{DO}$ and $\bullet_p^{PO}$ correspondingly:

$$\begin{gathered} \forall a \in \mathbf{M_1} \ (a \bullet_p^{PO} P = P \bullet_p^{PO} a = P) \\ \forall a \in \mathbf{M_1} \ (a \bullet_p^{DO} D = D \bullet_p^{DO} a = D) \end{gathered} \tag{4.7}$$

According to these properties, the magmas $\mathcal{M}_p^{PO} = \langle \mathbf{M_1}, \bullet_p^{PO}, N/A \rangle$ and $\mathcal{M}_p^{DO} = \langle \mathbf{M_1}, \bullet_p^{DO}, N/A \rangle$ are monoids. These monoids are not groups (but they are semigroups as was shown earlier), as they do not have inverse elements.

It should be noted that the presented formalisation of the deny- and permit-overrides combining introduces some modifications in comparison with the original algorithms defined in the considered XACML specification [153]. These modifications were introduced to resolve some counterintuitive

[6] It should be noted that natural orders for $\bullet_p^{PO}$ and $\bullet_p^{DO}$ on the set M_1 form two ordered sets that are, in turn, lattices defined in terms of the order theory.

cases of the 'Indeterminate' decision processing during the policy combining in XACML. According to the XACML algorithms, when an Indeterminate decision is combined with a Deny decision when the permit-overrides is used and with a Permit decision when the deny-overrides is used, a Deny decision is returned instead of the Indeterminate. These cases are counterintuitive, because when an Indeterminate decision is resolved into a Deny decision, it is interpreted by the PDP itself. However, according to the general conception of XACML, the interpretation of Not-applicable and Indeterminate decisions is a responsibility of the PEP. For this purpose, several types of PEPs were introduced that differ in how these decisions are processed. Moreover, these cases make the policy permit-overrides and deny-overrides combining operations asymmetric, while rule-combining algorithms do not have such anomalies and are symmetric. This can lead to the fact that decisions unexpected by the policy authors can be produced[7].

4.5.3 Policy and rule evaluation

A policy evaluation procedure is defined using the semantic function $evaluate^P : Policy \times REQ \rightarrow M_1$. The policy construct consists of a target and a rule combining constructs. The semantic expression for the semantic function $evaluate^P$ in Formula 4.8 is presented for the first production rule at row 4 in Figure 4.2. The policy evaluation is carried out using a procedure similar to the one for the policy set evaluation with the difference that rules are combined to produce a result decision, instead of policies or policy sets. Hence, the same auxiliary function P^e is used in the definition of the $evaluate^P$ function (see Formula 4.8) but the intermediate function f is required, as rules produce evaluation decisions from the set M_2 and, correspondingly, an element in this set is returned by the rule combining function $evaluate^{CombRule^{PO}}$ (if the permit-overrides is considered). The function $f : M_2 \rightarrow M_1$ maps decisions returned by the rule combining function into elements of the set M_1 before the function P^e is applied to them. This function maps coincident elements (P, D and N/A) to each other. The Indeterminate decisions $PInd$ and $DInd$, which designate that the Indeterminate decision was returned by a rule with the corresponding effect in its $Effect$ part, are mapped to the decision Ind. Accordingly, information about the rule's possible effect is eliminated in M_1, as this information is required only at the rule combining level.

$$evaluate^P([\![\texttt{CombPO Target RuleCombPO}]\!], req) = \tag{4.8}$$
$$P^e(evaluate^T([\![\texttt{Target}]\!], req), f(\ evaluate^{CombRule^{PO}}([\![\texttt{RuleCombPO}]\!], req))$$

The definition of the rule combining procedure, represented using the following semantic function: $evaluate^{CombRule^{PO}}$, is equivalent to the policy combining procedure specified using the function $evaluate^{CombPol^{PO}}$ (see Formula 4.2). The permit- and deny-overrides rule combining is

[7]Moreover, in the XACML specification version 3.0 [50] the mentioned inconsistencies were resolved and the updated versions for permit- and deny-overrides policy combining algorithms were specified.

Table 4.3: Table of values for Rule evaluation function R^e

Target \ Condition	$False$	Ind	$True$
$False$	N/A	N/A	N/A
Ind	$DInd$ or $PInd$	$DInd$ or $PInd$	$DInd$ or $PInd$
$True$	N/A	$DInd$ or $PInd$	D or P

also carried out using auxiliary binary operations: $\bullet_r^{PO}$ and $\bullet_r^{DO}$ (see Section 4.5.4). The evaluation of individual rule constructs is formalised using the semantic function $evaluate^{Rule} = Rule \times REQ$. The only production rule applicable to the non-terminal `Rule` substitutes it with the string `Effect Target BoolVal` (see the rule at row 7 in Figure 4.2). So a rule consists of an `Effect` part (can be Permit or Deny, see the rule at row 8 in Figure 4.2), a target and a condition. A semantic equation for the $evaluate^{Rule}$ function is represented in Formula 4.9 (for the case when a rule has `Permit` construct in its effect part). The decision from the `Effect` part is returned by the rule if the target and condition, which are Boolean values, are both evaluated to $True$. The evaluation of the target is represented using the semantic function $evaluate^T([\![\texttt{Target}]\!], req)$. When this function returns a $True$ value, this indicates that the rule is applicable and its condition should be evaluated. If the target is $False$, the rule is not applicable. If it returns Ind, $PInd$ (or $DInd$) is returned designating that Permit (or Deny) could be returned if an error did not occur. A complete table of values for the $R^e : EffectSpec \times TRVal \times TRVal \to M_2$ function, which defines the rule evaluation procedure, is presented in Table 4.3 (set $EffectSpec = \{Permit, Deny\}$).

$$evaluate^{Rule}([\![\texttt{Permit Target BoolVal}]\!], req) = R^e(P, evaluate^T([\![\texttt{Target}]\!], req), evaluate^{BoolVal}([\![\texttt{BoolVal}]\!], req) \quad (4.9)$$

4.5.4 Rule combining algebras

Rule combining algorithms are represented as binary operations $\bullet_r^{PO}$ and $\bullet_r^{DO}$ that combine two rule decisions from the set M_2 into a resulting decision also from the set M_2: $\bullet_r : M_2 \times M_2 \to M_2$. These operations were analysed, similarly to the policy combining operations, and are represented using an algebra $\mathcal{A}_r$:

$$\mathcal{A}_r = \langle \mathbf{M_2}, \{\bullet_r^{PO}, \bullet_r^{DO}, N/A, P, D\} \rangle \quad (4.10)$$

where

$\bullet_r^{PO}$ - is the binary operation for permit-overrides combining.

$\bullet_r^{DO}$ - is the binary operation for deny-overrides combining.

N/A, P and D - three special nullary operations representing designated elements.

Tables of values for the two operations are presented in Table 4.4. The permit-overrides com-

Table 4.4: Tables of values for permit- and deny-overrides rule combining operations $\bullet_r^{PO}$ and $\bullet_r^{DO}$

$\bullet_r^{PO}$	P	$PInd$	D	$DInd$	N/A
P	P	P	P	P	P
$PInd$	P	$PInd$	$PInd$	$PInd$	$PInd$
D	P	$PInd$	D	D	D
$DInd$	P	$PInd$	D	$DInd$	$DInd$
N/A	P	$PInd$	D	$DInd$	N/A

$\bullet_r^{DO}$	D	$DInd$	Permit	P(Ind)	N/A
D	D	D	D	D	D
$DInd$	D	$DInd$	$DInd$	$DInd$	$DInd$
P	D	$DInd$	P	P	P
$PInd$	D	$DInd$	P	$PInd$	$PInd$
N/A	D	$DInd$	P	$PInd$	N/A

bining algorithm gives priority to Permit, and the deny-overrides algorithm to Deny. As in rule combining it is possible to determine which decision could be returned instead of Ind, two indeterminate decisions have different priorities relative to the 'weaker' deterministic decision (i.e., Permit or Deny) in a current algorithm: Permit in deny-overrides and Deny in permit-overrides. For example, in permit-overrides, $PInd$ overpowers D because if a rule with the Permit effect had returned a Permit or Deny decision, instead of Ind, this decision would have priority over Deny.

Similarly to the policy combining operations, these rules combining operations are commutative, idempotent and associative. So the magmas $\mathcal{L}_r^{PO} = \langle \mathbf{M_2}, \bullet_r^{PO} \rangle$ and $\mathcal{L}_r^{DO} = \langle \mathbf{M_2}, \bullet_r^{DO} \rangle$ are semigroups and semilattices. As they are semilattices, natural orders for them can be defined (see Formula 4.11 with the permit-overrides order, Formula 4.12 with the deny-overrides order). As these orders are not duals, the algebra $\mathcal{L}_r = \langle \mathbf{M_2}, \bullet_r^{PO}, \bullet_r^{DO} \rangle$ is not a lattice.

$$N/A \leqslant^{PO} DInd \leqslant^{PO} D \leqslant^{PO} PInd \leqslant^{PO} P \quad (4.11)$$

$$N/A \geqslant^{DO} PInd \geqslant^{DO} P \geqslant^{DO} DInd \geqslant^{DO} D \quad (4.12)$$

The element N/A is an identity element for both operations $\bullet_r^{DO}$ and $\bullet_r^{PO}$, so magmas $\mathcal{M}_r^{PO} = \langle \mathbf{M_2}, \bullet_r^{PO}, N/A \rangle$ and $\mathcal{M}_r^{DO} = \langle \mathbf{M_2}, \bullet_r^{DO}, N/A \rangle$ are monoids (but not groups). Special elements P and D in the algebra $\mathcal{A}_r$ are adsorbing elements for the operations $\bullet_r^{PO}$ and $\bullet_r^{DO}$.

4.5.5 Target and condition evaluation

The part of the abstract syntax grammar presenting possible structures for targets and conditions in a XACML policy is shown in Figure 4.4. Both target and condition are specified by a policy author in a policy body and represent conditions that should be evaluated over a policy request and return a truth value. The target has a fixed structure that enables a fast retrieval of applicable policies from a large policy repository. The condition can represent any expression as compositions

of XACML functions, provided that these functions match in output/input data types. As XACML supports several data types and a large number of functions, operating with these data types, the grammar shows only a part of the XACML syntax for targets and conditions expressions, where only Boolean and integer data types are utilised.

(9) Target ::= Subjects Actions Resources Environments

(10) Subjects ::= Subject | Subject Subjects

(11) Subject ::= Matcher | Matcher Subject

(12) Matcher ::= MatchFunction TypedValue AttributeDesignator | MatchFunction TypedValue AttributeSelector

<Production rules (10) - (11) should be repeated for Action, Resource and Environment>

(13) MatchFunction ::= **IntEqualF** | **BoolAndF**

(14) TypedValue ::= **IntegerType** Integer | **BooleanType** Boolean

(15) AttributeDesignator ::= RequestSlot AttributeIdentifier DataType

(16) AttributeSelector ::= XPathExpression DataType

(17) RequestSlot ::= **Subject** | **Action** | **Resource** | **Environment**

(18) DataType ::= **IntegerType** | **BooleanType**

(19) Val ::= **IntAddF** Val_1 Val_2 | **IntOneAndOnlyF** BagVal | Integer

(20) BoolVal ::= **BoolAndF** $BoolVal_1$ $BoolVal_2$ | **IntEqualF** Val_1 Val_2 | **IntIsInF** Val BagVal | **IntSetEqualF** $BagVal_1$ $BagVal_2$ | Boolean

(21) BagVal ::= RequestSlot AttributeIdentifier **IntegerType** | XPathExpression **IntegerType** | **IntUnionF** $BagVal_1$ $BagVal_2$ | **IntBagF** Val_1 Val_2

(22) Integer ::= <Integer number> Boolean ::= **True** | **False**

Figure 4.4: Abstract syntax for XACML policies (cont.)

A fundamental complex data type in the evaluation of conditions and targets is a Bag of values ($BagVal$ in Figure 4.4). Bags can contain only elements with the same type and each element can occur several times in a bag. A bag of values can be defined formally as a multiset. A set of all possible multisets (including multisets with truth values) will be designated as $MSet$. A multiset, among other values, can contain indeterminate values (e.g., Ind^{Int}). When XACML constructs that retrieve values from a policy request are carried out, the retrieved values are represented as a bag of values. In XACML there two such constructs: an attribute designator or an attribute selector. The attribute designator construct is used to retrieve attributes from requests using their identifiers. The attribute designator has the following structure: `RequestSlot AttributeIdentifier DataType` (see row 15 in Figure 4.4). The first construct designates which category this attribute corresponds to: subject, resource, action or environment. The second construct is an attribute identifier and the last construct is a data type. In our model, we represent the execution of an attribute designator using the semantic function $evaluate^{Design} : AttributeDesignator \times REQ \rightarrow MSet$. The execution of attribute designators is not considered in detail and can be abstractly represented using the auxiliary function

$getValueDesign : Category \times Identifier \times Types \times REQ \rightarrow MSet$, where $Types$ is a set of all possible data type names. This function returns a bag of values with policy request attributes whose identifiers, categories and types match the designator. Alternatively, an empty bag can be returned. The attribute selector construct consists of an XPath expression and a data type (see rule at row 16 in Figure 4.4). When a selector is evaluated, this expression is evaluated over the XML document representing the decision request. The execution of an attribute selector is represented using the semantic function $evaluate^{Select} : AttributeSelector \times REQ \rightarrow MSet$ and formalised using the auxiliary function $getValueSelect$ which has the same signature as $getValueDesign$ but instead of the category and identifier for the identification of attributes it uses an XPath expression.

4.5.5.1 Target evaluation

Policy and rule targets consist of four levels (see rules at rows 9 - 12 in Figure 4.4). The overall target expression (represented as non-terminal `Target` at the grammar) contains four condition expressions, corresponding to the four categories in the policy request. As it is shown by Formula 4.13, the target matches a request if all these conditions are satisfied[8]. At the next level, conditions for each category of the request are specified as disjunctive condition expressions. One of these expressions should be satisfied over the data about the corresponding category in the request. For example, $evaluate^{Ss}([\![\texttt{Subject Subjects}]\!], req) = evaluate^{S}([\![\texttt{Subject}]\!], req) \vee evaluate^{Ss}([\![\texttt{Subjects}]\!], req)$. At the next level, each construct `Subject`, `Resource`, `Action`, `Environment` is specified as a conjunction of lower-level attribute matchers. In turn, each matcher defines one condition over one attribute value. For example, for the subject part of the target: $evaluate^{S}([\![\texttt{Matcher Subject}]\!], req) = evaluate^{Matcher}([\![\texttt{Matcher}]\!], req) \wedge evaluate^{S}([\![\texttt{Subject}]\!], req)$.

$$evaluate^{T}([\![\texttt{Subjects Actions Resources Environments}]\!], req) = p_1 \wedge p_2 \wedge p_3 \wedge p_4, \quad (4.13)$$
$$where\ p_1 = evaluate^{Ss}([\![\texttt{Subjects}]\!], req), \ldots, p_4 = evaluate^{Es}([\![\texttt{Environments}]\!], req)$$

Truth-value functions, which were used to specify target expressions, operate on the set $TRVal$, which also contains the Indeterminate value. These functions are defined using a method that is common in definitions of connectivities for three-valued logics: a correspondence of truth values $\langle True, Ind, False\rangle$ to numbers $\langle 1, 1/2, 0\rangle$ is defined. Then conjunction and disjunction are defined as min and max functions, negation - as the $1 - x$ function[9].

Matcher constructs are used at the lowest level of target expressions and can also be defined

[8] In the XACML specification, it is also required that the target should be evaluated to Ind whenever one of its constituent conditions (`Subjects`, `Resources`, `Actions` and `Environments`) has returned Ind (even if some condition has returned False). This requirement is specific to the highest level of the target evaluation and is not specified for truth-value functions or other levels of the target.

[9] This definition of connectivities corresponds, for example, to the three-valued logic of Łukasiewicz [65] $\mathbf{L}_3$

using a disjunction. A matcher construct consists of a function identifier `MatchFunction`, an attribute selector or designator and a constant value `TypedValue` that is specified directly in a policy (see rule at row 12 in Figure 4.4). As a `MatchFunction` construct can be used an identifier of a function with two input parameters that returns a boolean value. The type of its first argument should match the type of the constant `TypedValue` and the second argument's type - the designator's or selector's type. The matcher is evaluated to $True$ if the function `MatchFunction` returns true when it is applied to value `TypedValue` and one of the values in a bag retrieved using the designator or selector. An example of the matcher definition for the integer type and the 'equal' function is presented in Formula 4.14. The auxiliary function *equal* is used to compare integer values. The semantic function *valueInt* returns an integer from the set VAL corresponding to its syntactic specification `Integer` (see rule at row 22 in Figure 4.4).

$$
\begin{aligned}
evaluate^{Matcher}(\llbracket \texttt{IntEqualF IntegerType Integer AttributeDesignator} \rrbracket, req) = \\
if\ p == \emptyset\ then\ False\ else\ equal(k, p_1) \vee \cdots \vee equal(k, p_n),\ where \qquad (4.14) \\
p = evaluate^{Design}(\llbracket \texttt{AttributeDesignator} \rrbracket, req), p \equiv \{p_1, \ldots, p_n\}, k = valueInt(\llbracket \texttt{Integer} \rrbracket)
\end{aligned}
$$

4.5.5.2 Condition evaluation

Conditions in XACML are specified by policy authors as free-structure compositions of functions which should return a truth value in the set $TRVal$ as a result. The input values for the condition evaluation are bags of values retrieved using attribute designators and selectors and constant values specified within the condition structure by the policy author. In order to analyse all possible transformations of the input bags of values during the condition evaluation, before a truth value can be returned, all XACML functions should be analysed (i.e., functions described in the XACML specification [153]). First of all, these functions were classified according to their abstract signatures. In abstract signatures, only three types of values are distinguished: truth values in the set $TRVal$, all other atomic values in the set VAL and bags of values in the set $MSet$. Also if a function has several arguments with the same type, in its abstract signature all these arguments are represented using one argument with the additional '$^+$' suffix.

Based on this table, a chain of possible transformations that can be carried out with input bag values within conditions can be traced. The corresponding state chart is represented in Figure 4.5. All values are retrieved from the policy request as bags of values. After the execution of a function, they can be converted to a single value or a truth value. The truth value can be used in logical expressions or it can be returned as a result. The single value can be put into a bag (shown using the backward arrow in the chart, corresponding to the '*type*-bag' function) or it can be used in a function that also returns a truth value. In order to represent these transformations of values in the abstract syntax grammar, from each class of functions in Table 4.5 one concrete function

Table 4.5: Type of XACML functions according to their abstract signatures

No	Type of function	Examples
1.	$VAL^{+} \rightarrow VAL$	Integer-add (**IntAddF**), string-concatenate
2.	$TRVal^{+} \rightarrow TRVal$	All truth-value functions: or, not, and (**BoolAndF**)
3.	$VAL^{+} \rightarrow TRVal$	Integer-equal (**IntEqualF**), integer-less-than
4.	$MSet \rightarrow VAL$	Integer-one-and-only (**IntOneAndOnlyF**)[10]
5.	$VAL, MSet \rightarrow TRVal$	Integer-is-in (**IntIsInF**)[11]
6.	$VAL^{+} \rightarrow MSet$	Integer-bag (**IntBagF**)
7.	$MSet^{+} \rightarrow TRVal$	Integer-set-equals (**IntSetEqualF**)
8.	$MSet^{+} \rightarrow MSet$	Integer-intersection, integer-union (**IntUnionF**)

[10] Check that there is only one integer in the input bag and return this integer value.
[11] Return a bag of integers with all values contained in the input bag parameters.

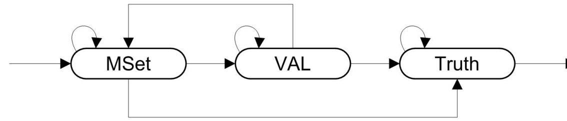

Figure 4.5: All possible transformations between abstract data types

was chosen and formalised as a production rule (see rules at rows 19 - 21 in Figure 4.4). For the reduction of the grammar size, only functions with integer and boolean arguments and values were chosen. Indeterminate values are returned if one of its arguments is an Indeterminate value, or an Indeterminate value is contained in an input bag, or an error occurred during the evaluation.

4.6 Obligations generation during policy evaluation

Obligations in XACML are specified as part of policies and policy sets. They are represented as sets of actions. Each action is represented using its identifier and a set of arguments. For each obligation, it is defined which decision, viz., Permit or Deny, this obligation should be returned with. When a policy or policy set where an obligation is specified is evaluated into a Permit or Deny decision and this decision matches a decision for the obligation, this obligation is returned along with the authorisation decision from this policy or policy set. Considering the policy combining algorithms described before, obligations propagate upwards in a hierarchical policy set structure. For example, if a policy set was evaluated to Permit due to a Permit decision returned by a constituent policy and this policy decision is accompanied with a set of obligations, these obligations should be attached to a resulting policy set decision along with the obligations specified within the current policy set. When several policies returned Permit decisions along with their obligations, all these obligations should be attached to the resulting decision. Policies that were not evaluated during the policy combining (e.g., when permit-overrides is used and the first policy has returned a Permit decision), correspondingly, do not return their obligations.

4.7 Conclusion

This chapter contains the description and the analysis of the XACML policy specification language which will be used in the next chapter as a tool for the policy specification in the policy-based planner. As part of this analysis, the following drawbacks were found: first, an absence of knowledge regarding which information should be included in a policy request being constructed in order to utilise it during the policy evaluation; second, an absence of the possibility to define as part of the obligations specification within a policy how these obligations should be executed relatively to each other and to the requested action (e.g., define an execution order or more complex relations between them). These drawbacks will be resolved during the policy-based planner design as part of the XACML policy language extension. The main contribution of this chapter is the introduced formalisation for the XACML policy language with the focus on the formalisation and analysis of the XACML policy evaluation algorithm. This formalisation uses as a basis the abstract syntax grammar introduced in this chapter. Using this grammar, mappings from the XACML syntactic constructs to objects that represent the process of their evaluation formally were defined. As part of this formalisation, the XACML policy and rule combining algorithms were formalised as abstract algebras and their properties were analysed. This formalisation is utilised in Chapter 6 where an extension of the policy-based planner, viz., the postponed policy enforcement mechanism, is introduced. For the realisation of this mechanism, the XACML policy language and its evaluation algorithm were extended with the possibility to evaluate policies in situations when not all relevant information is available. The formalisation provided the means to formally introduce the extension of the XACML policy evaluation algorithm and demonstrate its required properties, including its relations with the original version of the algorithm.

Chapter 5

Policy-based planner

Objectives:

- *Introduce the main components of the policy-based planner and the main interaction processes between them, which underlie the functioning of the policy-based planner.*
- *Describe the main elements used for the specification of the planning environment and the algorithms for their processing.*
- *Introduce an adaptive policy requests construction technique for generation of policy requests containing only information that can be required during the policy evaluation.*

5.1 Introduction

Planning operates with a model of the system and produces a plan that should be executed within this system in order to achieve a goal. Policies specify regulations that determine how actions should be carried out in the system. A policy-based planner that combines planning technology and a policy-based approach should enforce policies during the planning. Thus, it should guarantee that the resulting plan is conformant with all policies that are applicable to it.

Using a policy-based planner, it is possible to apply planning techniques in heterogeneous environments where different regulations exist that are applicable only for specific parts of the environment and that control different aspects of processes carried out in that environment. These regulations can originate from different sources and can be specified by different persons independently. In the policy-based planning, such regulations can be specified using a policy specification language providing the means to combine simple policies specified by different authors into more complex policies and resolve conflicts between them using combining algorithms. Therefore, in the policy-based planner, policies are used to extend the functionality provided using the core HTN planning constructs. Using policies, it is possible to specify additional conditions on the execution of operators and methods. Additionally, using a policy obligations mechanism, it is possible to

extend core decomposition methods with additional tasks and actions in order to construct routines that satisfy requirements specific to the situation when it should be executed, for instance, specific to a domain of the planning environment. Moreover, the enforcement of policy obligations, which were generated based on the policies specified, is controlled using the obligations validation mechanism. It is checked that the obligations do not contradict with the overall principles of the planning environment and with the rules specified by other policy authors. The specification of regulations using policies provides the possibility to flexibly modify them. First of all, it is possible to specify policies that are valid only during a specific time interval. So the planning mechanism that produces plans with actions that should be executed at a specific moment in the future takes into account only relevant policies. Moreover, as the policies are specified in an external repository, they can be updated at run-time of the planner, independently from the core planning environment updates.

The planning algorithm of the policy-based planner is domain-independent. In order to carry out planning in a specific problem area, it should be provided with a specially designed planning environment specification. This approach is widely used in the planning community as it provides the means to join the efforts of planning specialists working in different problem areas and re-use (or adapt) the designed problem-independent techniques in different problem areas [116]. Correspondingly, the policy-based planner was designed based on the domain-independent HTN technique, extending it with mechanisms required for planning in environments with heterogeneous regulations. In turn, these mechanisms were also designed according to the domain-independent principles. The resultant policy-based planner can be utilised in different problem domains, where there are premises for its usage (i.e., planning should be carried out in environments with heterogeneous regulations).

In spite of the fact that the policy-based planner is a domain-independent engine, its design is based on its future utilisation within the CEP generation framework and, correspondingly, on the requirements specified based on the analysis of the student mobility area (see Chapter 3). First of all, the policy-based planner supports hierarchical planning. Arguments for the utilisation of hierarchical planning technology for the student mobility programmes development were stated in Chapter 2. Additionally, the utilisation of the hierarchical planning technology enables the fulfilment of some requirements specified for the CEP development solution (see Chapter 3). In concrete, student mobility scenarios can be modelled and generated during the planning using decomposition methods, supported by the HTN planning technology used. Next, the policy-based planner supports planning in environments where different terms and units are adopted in different domains for description of the same or related notions. In the policy-based planner, such terms and units can be coherently used since transformation rules are utilised to determine their relations and to convert them.

This chapter is organised as follows. An overview of the policy-based planner and its conceptual model are presented in Section 5.2. In this section, the main functions and interactions of the planner components are described. Next, each main component of the planner, in concrete, a planning engine, a policy engine and a transformation rules engine, is described, respectively, in Sections 5.3, 5.4 and 5.5. In the final part of the chapter, an adaptive policy requests generation technique that provides the means to generate policy request containing only information that can be required during the policy evaluation is presented (see Section 5.6).

5.2 Policy-based planner overview and general processes

A high-level architecture of the policy-based planning system is represented in Figure 5.1. Within this system, three engines are used, namely, a planning engine, a policy engine and a transformation rules engine. The planning engine contains the planner's world state model, which is used to represent a model of the environment where a plan will be carried out. This model is dynamically updated by the planner as a result of the execution of actions. These actions form a plan that is simulated within the planner's environment. When the plan is created, it should be carried out in a real environment in order to achieve a planning goal. The core planning algorithm used by the planning engine is domain-independent. For the operation within a specific problem domain, it utilises specifications of this planning domain containing descriptions of actions, tasks and decomposition methods, which can be used in this domain[1]. These specifications describe general principles and mechanisms of the operation within some problem area which are devised by a problem domain expert and are usually invariable in time. Within the policy-based planner, the planning engine also provides the main problem-solving interface: it receives descriptions of planning problems and provides plans generated for these problems.

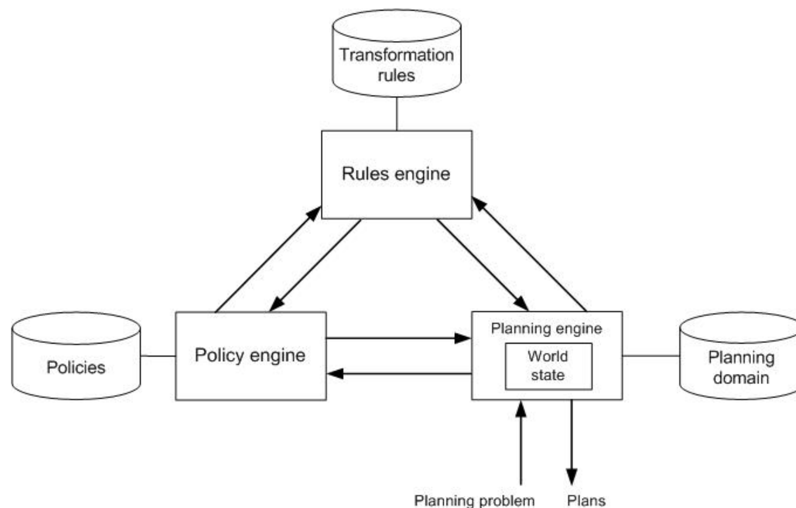

Figure 5.1: Overview schema of the policy-based planner

[1] That is, the planning domain performs the role of the process library.

The policy engine is used to evaluate policies during the planning. Policies are utilised to specify requirements that are valid only within a specific context, for example, a specific time interval, a specific domain within the planning environment or specific properties of the action being executed. Policies, as opposed to the planning domain, originate from different sources and can be specified by different authors. Policies can be updated dynamically during the system run. Moreover, policy can have a validity period, which determines the time interval during which the policy is active (it should be considered that the planning simulates the execution of actions taking into account their durations). The policy engine receives policy decision requests from the planning engine with descriptions of the action that should be executed within the planner's world state and all required information related to it. After the evaluation of these policy requests, the policy engine returns an evaluation outcome to the planning engine that enforces it.

The third main component of the policy-based planner is the transformation rules engine. In planning environments that cover different areas (e.g., geographical or organisational), different terms and units can be used to designate the same or related notions, for example, different terms can be used in different areas to denote the same object or characteristic. In order to interchangeably use these notions and units, transformation rules are used to define relations between them. If within some policy or a planning domain specification a term or a value at a specific scale is required, the policy or planning engine forms a request that is processed based on available transformation rules by the transformation rules engine.

5.2.1 Conceptual model

The conceptual model of the policy-based planner describes the main constructs used in policy-based planning and their relations. It consists of the following components:

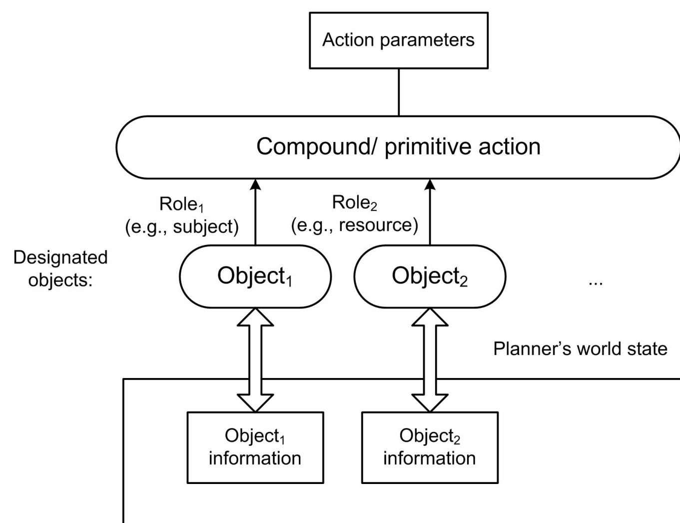

Figure 5.2: Conceptual model for action

Primitive action is the central notion of the policy-based planning. Primitive actions model modifications of the planner's world state. Thereby, they are used as constructs for the output plans development. Each action is related with one or several objects that have distinct roles in its execution (they are its designated objects). Relations between notions associated to the primitive action are presented in Figure 5.2.

Object represents an entity within the planning environment. Each object is stored in the planner's world state and is described by its properties and relations with other objects. Objects can be used in actions as designated objects with specific roles. Properties and relations of an object can be modified, as well as new objects can be saved into the planner's world state during the planning.

Role describes how a designated object is participating in an action. For different actions, different sets of roles can be used to designate specific objects of these actions. For example, a role can be 'subject', 'resource' or 'destination'.

Policy. Policies are statements that define requirements to processes being carried out within the planning environment. Policies can specify which actions can be executed and under which circumstances and prescribe specific routines for the execution of certain processes. In order to infer a policy decision, policy conditions can use any information about the action, including information from the planner's world state directly or indirectly related to its designated objects.

Time in policy-based planning is a variable that is maintained by the planner and represents time being modelled during the planning. When an action is executed, the planner increases a value of time variable with the length of time interval specified as the action's duration. As policies can be dynamically changed, they can be valid only during specific time intervals.

Compound tasks are tasks within the planning environment that are not executed directly, like actions, but they are decomposed into a partially ordered set of lower-level tasks and actions using the methods, designed as part of the planning domain. During the planning, compound tasks should be fully decomposed into primitive actions.

Compound action is a construct that shares the properties of a compound task and a primitive action. Compound actions are decomposed into lower level tasks and actions, are used to model modifications of the planner's world state and are used to construct the resulting plan. Compound actions, as well as primitive actions, are also related with corresponding sets of designated object with specific roles.

Obligations are actions or compound tasks that, according to policies, should be executed during the planning along with the current planning action. Obligations are triggered during the evaluation of policies against the policy decision request that contains the specification of the current action being requested.

Condition is a statement that should be evaluated at a certain point of the plan execution but that cannot be evaluated during the planning, because of the information missing at the planning stage. Further execution of the plan is possible only if all corresponding conditions are satisfied.

Plan specifies how a process designed during the planning should be executed. A plan in policy-based planning consists of actions connected with sequential and hierarchical relations and conditions, which should be satisfied during the execution of these actions. So plans have hierarchical structure that provides additional information about the internal organisation of the designed process and provides the means to model the process at different levels of abstraction.

Transformation rule is a rule that defines how to transform or relate a property of an object specified using terms and units adopted in one domain or classification system to another.

5.2.2 Main interaction processes

This section describes the main mechanisms designed for the interaction of the main components of the planner. These processes enable evaluation of policies during the planning, construction of conditional plans and extension of core planning domain decomposition methods using obligations that are specified in policies.

5.2.2.1 Actions legitimacy and policy evaluation

The specification of environment that is utilised by the policy-based planner can be defined as a tuple $Env = \langle PlanD, TransRule, PolicySpec \rangle$, where $PlanD$ is a specification of the planning domain containing definitions of actions, compound tasks and methods for their execution, $TransRule$ is a specification of transformation rules and $PolicySpec$ is a specification for policies that exist in this environment.

The policy-based planner is based on the classical model of an action-based planning system (see Chapter 2). In this model, in order to execute an action during the planning it should be *applicable* to the current planner's world state. A state-transition model of the system that the planner operates with is defined within the planning domain specification $PlanD$ (see Section 5.3). In concrete, a state-transition function *Apply* maps a state and an action to a new state that the system transfers to when this action is executed in this state. This function is defined only for pairs 'action-state' where the action is applicable to the state[2].

In policy-based planning, it is additionally required that each action that is executed by the planner and is included in a plan should be conformant with the policies $PolicySpec$. So in addition to the notion of action applicability, the *legitimacy* of action is introduced. Descriptions

[2]In this chapter, the term 'action' refers to both primitive and compound actions. Primitive actions are specified using operators in the planning domain. Compound actions are specified using compound action decomposition methods. The function *Apply* is defined in Section 5.3.

Table 5.1: Interpretation of policy decisions by the planning engine

Decision	Planner's reaction
'Permit'	execute
'Not applicable'	execute
'Deny'	backtrack
'Indeterminate'	backtrack

of designated objects that an action can refer to are stored in the planner's world state. Hence, policies are evaluated for an action in relation with the planner's world state where it is executed. A policy decision tuple $\langle d, OblStr, C\rangle$ is produced as a result of the evaluation of policies for action a in state s (see Formula 5.1). It contains a policy decision d ($d \in M_1 = \{P, D,\ N/A,\ Ind\}$). The tuple *OblStr* contains obligations Obl_i that should be executed during the planning. The tuple C contains conditions $Cond_j$ that should be added into the resulting plan and evaluated during the plan execution. The outcome of the policy evaluation for an action a and a state s is determined by the policies $PolicySpec$.

$$\langle s, a\rangle \Rightarrow^{PolEval} \langle d, OblStr, C\rangle \tag{5.1}$$

When the planning engine enforces results of the policy evaluation, it processes policy decisions according to Table 5.1. It executes actions that were evaluated to Permit and Not Applicable and blocks actions for which Indeterminate or Deny were returned. A Not Applicable decision designates that policies were not specified for this action-state combination. Since policies specify regulations that are supplementary for the core planning domain specification, it is assumed that all actions are permitted by default and policies limit possibilities of their application. An Indeterminate decision designates that an error occurred during the policy evaluation. When an Indeterminate decision is returned, the action is blocked due to the fact that the action could be evaluated into a Deny decision if the error had not occurred.

Definition 5.1. Legitimacy. Action a is legitimate for the execution in state s if policy decision tuple $\langle d,\ OblStr,\ C\rangle$ was produced during the policy evaluation for the action a and the state s and $d \in \{P, N/A\}$ □

Obligations that are contained in the tuple *OblStr* represent actions and compound tasks that should be executed along with the action being evaluated. The actions returned in the *OblStr* tuple should also be applicable to the corresponding planner's world states and should be legitimate. Therefore, during the planning, the legitimacy of actions is evaluated in a recursive manner. This recursive process is stopped when no obligations are returned for an action as a result of the policy evaluation.

The evaluation of policies for an action that should be executed in a current planner's world state is divided into two stages. During the first stage, a policy request *Req* is generated based on the current planner's world state and the action that should be executed (see Formula 5.2). Then,

this policy request is passed to the policy engine that evaluates it using policies specified in its policy repository (see Formula 5.3).

$$\langle s, a\rangle \Rightarrow^{Gen_R} Req \tag{5.2}$$

$$Req \Rightarrow^{Eval_R} \langle d, OblStr, C\rangle \tag{5.3}$$

The policy request contains information about the action and the relevant information from the current planner's world state in a form that can be processed by the policy engine. It contains information about the action's designated objects, action parameters and time interval for the action execution. Information from the current planner' world state is retrieved only if it could be used during the policy evaluation. The structure of the policy request is described in Section 5.4.1. A technique for the selection and transformation of information from the planner's world state for the policy request generation is described in Section 5.6. The constructed policy request is evaluated using the XACML policy evaluation algorithm that was formalised in Chapter 4.

5.2.2.2 Extensions of planning domain using policy obligations

Obligations are compound tasks and actions that should be executed during the planning along with some planning action. The obligations, which should be executed, are specified in *OblStr* tuple of the policy decision tuple produced during the evaluation of policies for this action. When policy authors specify policies *PolicySpec*, they define which obligations should be produced for specific policy requests. Details about the specification of obligations are given in Section 5.4.

Obligations in the *OblStr* tuple are divided into before-, during- and after-obligations according to the position where these obligations should be executed relatively to the action being evaluated:

$$OblStr = \langle Obl_B, Obl_D, Obl_A\rangle \tag{5.4}$$

where $Obl_y = \{\langle Obl_{11}, \dots, Obl_{1n}\rangle, \dots, \langle Obl_{m1}, \dots, Obl_{mk}\rangle\}$, $y \in \{B, D, A\}$ and Obl_i - the compound task or action that should be executed. Obligations organised in tuples should be executed in the order that corresponds to their sequence order in the tuple. Compound tasks and actions that are returned as obligations should be processed by the planner using the ordinary procedure that is used for the execution of other compound tasks and actions during the planning. Obligations can be executed in any order that respects their position relatively to the action for which they were returned and the sequence orders specified using tuples. If an obligation cannot be executed by the planner, the original action that it was produced for should be backtracked. If all obligations were successfully executed by the planner, actions that were generated as obligations and that were produced during the decomposition of obligations are added into the resulting plan. When during-obligations are processed, they decompose the action that they were produced for,

similarly as compound actions are decomposed using methods (during-obligations can be returned only for compound actions). Further details about the obligations processing are described in Section 5.3.

Using obligations, policy authors can introduce additional actions and compound tasks into processes modelled by the planner. As policies can be specified by different persons and be applicable only in specific context, this provides the possibility to flexibly adapt core processes specified within the planning domain during the planning according to different requirements specified by policies. Hence, in policy-based planning two sources of new tasks and actions exist: decomposition of compound tasks and actions by the planning engine and generation of obligations during the policy evaluation. Since using obligations different policy authors can participate in generation of new tasks during the planning, there is a need to ensure that a set of obligations generated can be executed in current situation during the planning (e.g., they do not contradict with each other) and that policy authors that have specified these obligations have rights for triggering the generated obligation-action or obligation-task. In order to control these issues, the obligations validation mechanism was introduced (see Section 5.4.3). Obligations validation rules can be specified that determine eligible combinations of obligations that can be returned during the evaluation of policies for a specific action. Obligations validation rules can be specified on a global level, that is, for the whole planning environment, as well as, for a specific policy. In the latter case, they limit a set of obligations that can be produced in specific situation by the author of this policy.

5.2.2.3 Conditional plans construction using policy conditions

In policy-based planning, the policy enforcement is performed on a restricted model of the system specified using the planning domain. This model is deterministic and represents only information that is known at the planning stage. Concrete outcomes of actions, which will be received during the execution of the plan, could be different from outcomes expected by the planner or can be unavailable during the planning. However, using policies, it should be possible to specify any regulations, including the regulations that operate with information available only during the execution of the plan.

Policies specified for the policy-based planner, in addition to constraints that should be enforced during the planning, can determine constraints that should be enforced during the execution of a plan. These conditions are returned in tuple C along with policy decisions as a result of the policy evaluation. They are attached to the corresponding action and are saved into the resulting plan. Conditions in tuple C are divided into before, after and during-conditions. When some condition is evaluated after the execution of an action, concrete outcomes of the action can be evaluated. If this condition is not satisfied, further execution of the plan can be forbidden. During-conditions

should be satisfied during the entire period of the action execution.

$$C = \langle C_B, C_D, C_A \rangle \tag{5.5}$$
$$C_x = \{\dots, Cond_i, \dots\}, x \in \{B, D, A\}$$

where C_B - before-conditions set, C_A - after-conditions set, C_D - during-conditions set, $Cond_i$ - individual condition statement. The specification of conditions in policies is described in Section 5.4.2.1.

Conditions returned within the resulting plan can be utilised by a controller that executes this plan or they can be analysed by a user in order to understand concrete requirements associated with the plan (for example, this can be used to choose one plan from available alternatives). In an educational area, conditions can be used to specify requirements on the learning outcomes of students. For example, an after-condition for the action '*pass_exam*' can be a minimum requirement on a mark that the student should get.

5.3 Planning

This section describes components that are used by the planning engine within the policy-based planner, including the constructs for the specification of the planning domain for a concrete problem area. The main planning algorithm implemented by this engine is also presented in this section. The planning engine of the policy-based planner utilises the HTN planning technology. Existing HTN planning technology described in the literature [116, 118] was taken as a basis for the planning engine design and was substantially extended with the object model of the planner's world state, policy request initiation mechanism, compound actions and corresponding methods, conditions and obligations mechanisms, support for hierarchical plans[3]. Procedures for processing of these constructs were added to the planning algorithm as presented in Section 5.3.5.

5.3.1 The planner's world state and its object model

As is commonly done in the planning community, the planner's world state is defined as a set of ground positive literals: $L = p(\tau_1^c, \dots, \tau_n^c)$, where p is a predicate symbol, $p \in Pred$, and τ^c is a term-constant, $\tau^c \in Term^c$. $Pred$ and $Term^c$ are respectively the sets of all possible predicate symbols and term-constants that can be used within the planner's world model. The set of all possible planner's world states is designated by $S = \langle s_0, \dots, s_n \rangle$. In the policy-based planner, in order to introduce the additional structuring of the planner's world state and have a possibility to reason about the planner's world state at a higher level of abstraction, an object model of the planner's world state is defined. For this purpose, all term-constants used in the

[3]Correspondingly, task network structures, basic operators, consisting of preconditions and effects, and basic methods, consisting of preconditions and task networks, and their processing principles were adopted.

planner's world state are divided into two disjoint sets: objects $Term^{Obj}$ and properties $Term^{Prop}$ ($Term^{Prop} \cap Term^{Obj} = \emptyset$, $Term^{Prop} \cup Term^{Obj} = Term^c$). Object-terms are used as object identifiers, they will be denoted as Obj_{ID}, $Obj_{ID} \in Term^{Obj}$. Property-terms $\tau^{Prop} \in Term^{Prop}$ are used to describe these objects, that is, specify their properties. Objects represent entities within the planner's domain. Using the division of term-constants into object-terms and property-terms, information about objects stored in the planner's world state can be identified and grouped together.

In the planner's world state, objects-terms are specified using literals with the reserved predicate symbol **Object**. For each object, its type is specified. A definition of object in the planner's world state is as follows:

$$\textbf{Object}\ (Obj_{ID},\ Obj_{Type}) \tag{5.6}$$

where Obj_{ID} is an object-term and Obj_{Type} is a special property-term, representing the object's type ($Obj_{Type} \in Term^{Prop}_{OType} \subset Term^{Prop}$)[4]. All other terms that are not marked using the **Object** literal are property-terms. All literals that are used in the planner's world state are divided into three disjoint sets: property-literals, relation-literals and fictitious literals[5]. *Property-literals* are literals that contain one only object-term. *Relation-literals* are literals that contain more than one object-terms. *Fictitious literals* are literals that contain no object terms[6]:

$$\text{Property-literal } L = p(\tau_1^c, \ldots, \tau_n^c), \text{ such that } \exists! \tau_i^c, \tau_i^c \in Term^{Obj} \tag{5.7}$$

$$\text{Relation-literal } L = p(\tau_1^c, \ldots, \tau_n^c), \text{ such that } \exists \tau_i^c, \tau_j^c, \tau_i^c \in Term^{Obj} \land \tau_j^c \in Term^{Obj} \land i \neq j \tag{5.8}$$

$$\text{Fictitious literal } L = p(\tau_1^c, \ldots, \tau_n^c), \text{ such that } \overline{\exists} \tau_i^c, \tau_i^c \in Term^{Obj} \tag{5.9}$$

As was defined in the conceptual model in Section 5.2.1, objects are related with actions using roles. Correspondingly, object-terms representing these objects are used as action parameters. The introduced object model constructs are used to retrieve information related to these objects and present it in the policy request for analysis during the policy evaluation. An algorithm of relevant information selection and its transformation into a policy request is described in Section 5.6.

In addition to term-constants, within the planning domain specification variables are used. Identifiers of variables contain prefix '?' (e.g., ?*Student*). During the planning, variables can instantiated by any term-constant: either an object-term, or a property-term. A set of all variables used in a planning domain specification is denoted as $Term^v$. As defined, the planner's world state contains only ground literals, so variables cannot be used within it.

[4]Type for an object $Type(Obj_{ID})$ is returned using function $Type(Obj_{ID})$.

[5]Here and in the following description, it is assumed that all literals are positive, as it is required for literals in the planner's world state. So this will not be stated explicitly.

[6]It is possible to use fictitious literals in the planner's world state, but they should be used only for storing required auxiliary information, e.g., time. Fictitious literals cannot be used during the policy evaluation, because as they are not related to any objects, their relations with actions that are being evaluated can not be determined

The policy-based planner constructs plans that span throughout specific time intervals. Hence, since policies are dynamic, they can be changed during these time intervals. Such policy changes should be modelled (if they are known in advance), and during the planning only policies that are in force at the current time should be enforced. Therefore, the planner's world state should contain information about the current time. This information should be used to determine time points and intervals when actions are performed.

Different structures can be used to represent time (e.g., relative time can be represented using real, natural numbers or absolute time can be represented as calendar dates). In the described policy-based planner, absolute time values are adopted. Time is represented using 3 values: day of the month, month and year. They are saved into the planner's world state using reserved predicate symbols **CurDay**, **CurMonth** and **CurYear**, respectively (e.g., **CurYear**(2014)). This provides the possibility to specify policies that are valid only during specific calendar intervals[7] or that are valid periodically (e.g., in a specific month). Values of the time variables are used as reference values that indicate a time interval when the action starts and finishes. As this time intervals are coarsely specified, that is, the minimum time unit is one day, several states of the planner's world state model can correspond to the same minimum time unit. Therefore, actions can start and terminate at the same time unit (i.e., on the same date).

5.3.2 Planning domain specification

In the policy-based planner, the planning domain $PlanD$ is defined as a tuple $\langle O, MethCA, MethCT \rangle$, where O is a set of operators that model the execution of primitive actions within the planner's world state model, $MethCA$ is the set of compound actions decomposition methods and $MethCT$ is the set of compound tasks decomposition methods for modelling, respectively, the execution of compound actions and tasks.

5.3.2.1 Tasks

Three types of tasks are supported by the planner: compound tasks, primitive actions and compound actions. Each task is represented as a task atom $TA = TA^S(\tau_1, \ldots, \tau_n)$, where TA^S is a task symbol and $\tau_1, \ldots, \tau_n$ are terms that are used as parameters for this task such that $\tau \in Term^c \cup Term^v$ (task atom parameters can be both term-constants and term-variables). Task symbols for primitive actions should begin with an exclamation mark '!'; compound actions should begin with an ampersand sign '&'. *Primitive actions* are atomic and they are directly executed within the planner's world state model resulting in its update. *Compound tasks* are not executed directly, they are decomposed using compound tasks decomposition methods into lower level task networks. *Compound actions* unite compound tasks and primitive actions features: they are de-

[7]This is important when using the planner the processes that are managed based on policies specified in terms of calendar dates are modelled, e.g., HE regulations.

composed using compound actions decomposition methods that, in addition to decomposing the task, modify the planner's world state. The utilisation of the primitive actions and compound actions during the planning should conform to policies. When the planning engine is going to apply a primitive or compound action, the policy decision request should be generated and evaluated using the policy engine.

When a compound or primitive action is executed, their start and end time points should be determined. The start and end time points of an action can be different or equal, indicating that the action is carried out within the same day. The interval between start and end time points of an action will be referred as its execution interval. It is assumed that modifications of the planner's world state that the action brings are carried out at the moment when the action is complete, that is, at the end time point of the action. When an action is executed, the current time values in the planner's world state should be updated (for instant actions, these time values will not be changed). During the execution of a compound action, other actions can be produced as a result of its decomposition. Correspondingly, these actions should be executed in a nested manner. In this case, the nested actions should be executed within the time interval of the compound action that has produced it.

Another important construct that is used in HTN planning for the specification of planning problems and decomposition methods is a *task network* (TN), that is, a partially ordered set of tasks. We adopt the hierarchical representation of task networks where a task network is defined as a nested structure:

$$TN = \langle TN_1, \ldots, TN_n \rangle \mid \{TN_1, \ldots, TN_m\} \mid TA \tag{5.10}$$

$\langle TN_1, \ldots, TN_n \rangle$ is an ordered task network where tasks should be executed only in the order specified and $\{TN_1, \ldots, TN_m\}$ is an unordered task network where tasks can be executed in any order.

5.3.2.2 Operators

Within the planner's world state model, primitive actions are executed using operators. In the policy-based planning, when an action is executed, in addition to its applicability, its legitimacy should be checked. So the definition of operator, in addition to an expression for the evaluation of action's applicability, includes a structure with values for the policy request generation. This policy request is evaluated to check the operator's legitimacy.

Definition 5.2. Operator o is a construct that defines a primitive action execution procedure. An operator is defined as a tuple:

$$o = \langle task(o), precondition(o), duration(o), policyPar(o), effect^{+}(o), effect^{-}(o) \rangle \tag{5.11}$$

where $task(o) = TA_i$ is the operator's head, the primitive action task atom that this operator can be applied to, $precondition(o)$ is the precondition expression, $duration(o)$ is the duration expression, $policyPar(o)$ - the policy parameters tuple, $effect^+(o)$ and $effect^-(o)$ are, respectively, positive and negative effects of the operator□

Operators are specified within the planning domain using operator schemas that define all constituents of the operator. Operator schemas, in addition to constant terms, can contain variables (as in [116]). An operator is relevant to a ground primitive action task atom TA and can be applied to it if there is a substitution for all variables in $task(o)$ such that $task(o)$ is equal to TA. The predicate symbol used in a primitive action task atom determines the operator that can be used to execute it. Therefore, one predicate symbol can be used only in one operator schema within the planning domain. The operator's precondition $precondition(o)$ is used to determine if the operator is applicable in the current planner's world state. Preconditions are specified as expressions that can be evaluated as true or false in a planner's world state[8]. Operator o is *applicable* if its precondition expression is satisfied in the current planner's world state (i.e., there is a substitution of its variables such that the expression becomes a consequent of the literals in the planner's world state). $duration(o)$ is an expression where values for three time variables $?DEnd$, $?MEnd$, $?YEnd$ that represent the end time point of the action are assigned.

The policy parameters tuple $policyPar(o)$ is specified within the operator schema as a tuple that defines parameters for the construction of policy request representing this operator:

$$policyPar(o) = \langle ObjSet, ParamSet \rangle \tag{5.12}$$

where the objects set $ObjSet = \{\langle Obj_{ID_1}, Role_1\rangle, \dots \langle Obj_{ID_n}, Role_n\rangle\}$ defines this action's designated objects Obj_{ID_j} along with their roles $Role_j$. Objects contained in the objects set will be used in the policy request as designated objects. The parameters set $ParamSet = \{\langle AParVal_1,$ $AParName_1\rangle, \dots \langle AParVal_m, AParName_m\rangle\}$ is a set representing attributes of the action that will be contained in the policy request. Each attribute is defined as $AParName_i$ and $AParVal_i$, the name of the attribute and its value, respectively. In operator schemas, designated objects Obj_{ID_j} are specified as variables that are instantiated with object-terms during the application of the operator. Attribute values $AParVal_i$ can be either term-constants, or variables that are instantiated with term-constants. For these variables, only variables used within the operator's head can be used. The main aim of the policy parameters tuple is to specify how concrete parameters used in the operator's head should be utilised during the policy request generation. Each desig-

[8] As opposed to policies, which are specified using the XACML policy language, planning preconditions do not support other truth values than true and false. When an error occurs during the precondition evaluation, the whole precondition becomes unsatisfied. The mechanism for the precondition specification and evaluation was adopted from an existing HTN planner (see Chapter 8). The precondition expressions are specified as literals, functions and variable assignments, connected using negation, conjunction and disjunction connectives, and additionally can utilise universal quantifiers.

nated object in the objects set is represented within the policy request by a distinct component containing all information about this object that can be required during the policy evaluation. For this purpose, the '*subject* - *action* - *resource* - *environment*' model utilised in XACML was extended in order to represent within the policy request all information about any number of the designated objects (see Section 5.4).

Each operator schema represents a set of operator instances that are derived when variables in the operator's head are instantiated. Such operator instances can be applied during the planning for execution of tasks atoms that represent primitive actions. It should be noted that values used in the operator's head should determine uniquely the substitution that is used to satisfy the operator's precondition and the action duration (therefore, they also determine uniquely the operator's effects). Further, it is assumed that an operator designated as o can represents only an operator instance.

Constructs $effect^{+}(o)$ and $effect^{-}(o)$ represent positive and negative effects of the operator, specified as sets of literals. When an operator is executed, the planner's world state is updated, such that literals contained in $effect^{-}(o)$ are removed from the current state and literals in $effect^{+}(o)$ are added to the current state. Additionally, along with the application of the operator's effects, current time values stored within the planner's world state using literals **CurDay**(τ_1^c), **CurMonth**(τ_2^c), **CurYear**(τ_3^c) are deleted and new time values **CurDay**$(?DEnd)$, **CurMonth**$(?MEnd)$, **CurYear**$(?YEnd)$ are added. The planner's world state updates are represented using the function $Apply^{op} : S \times O \rightarrow S$, where O is the set containing all possible operator instances within the planning domain.

Based on the operator instance being applied during the planning, a policy vector is generated that contains all required information about this operator instance in order to build a policy request. The policy vector $polVec(o) = PolVec$ is derived from the operator instance. The policy vector is represented as a tuple:

$$PolVec = \langle ObjSet, TA^{S}, ParamSet, TInterval \rangle \tag{5.13}$$

So the policy vector extends the policy parameters tuple $policyPar(o)$ with values TA^{S} and $TInterval$. TA_i^{S} is the operator's head task symbol, which is used as an action in the policy request. $TInterval = \langle ActBeg, ActEnd \rangle$ are parameters identifying the begin and end time points for this action. They are specified as tuples $\langle Day, Month, Year \rangle$. Values in the $ActBeg$ tuple are equal to the current time values and values in the $ActEnd$ are determined using the $duration(o)$ expression. The policy vector together with the planner's world state contains all required information for generation of the policy request. An operator instance is *legitimate* if a policy request generated based on its policy vector and the planner's world state before its execution (designed

as $Req = Gen_R(s, polVec(o)))$ was evaluated using the policy engine into a policy decision tuple $\langle d, OblStr, C\rangle$ (designated as $Eval_R(Req) = \langle d, OblStr, C\rangle$) where d is a policy decision such that $d \in \{P, N/A\}$. The overall policy evaluation process for an operator instance o in a state s is designated as $PolEval(s, polVec(o)) = \langle d, OblStr, C\rangle$. The process of policy request generation is described in Sections 5.4.1 and 5.6.

The following is an example of policy parameters tuple specification and generation of a policy vector for an operator schema with operator's head $!recognise(?Student, ?Mod_1, ?Mod_2, ?Aim)$. The corresponding primitive action task atom designates an action when the module $?Mod_1$ is recognised as equivalent to the module $?Mod_2$ for the student $?Student$. The recognition is carried out with the aim $?Aim$, which can be a graduation, a transfer or a prerequisites evaluation. In the operator schema, these 4 variables are allocated in the policy parameters tuple according to the following structure: $\langle\{\langle ?Student,\ Subject\rangle, \langle ?Mod_1,\ ModuleRecognise\rangle, \langle ?Mod_2,$ $ModuleSupport\rangle\},\ \{\langle ?Aim,\ AimOfRecognition\rangle\}\rangle$. This designates that in the policy request the student should be used as a designated object with the $Subject$ role, the first module parameter - as a designated object with the role $ModuleRecognise$ and the second module parameter - as a designated object with the role $ModuleSupport$. The last parameter of the action is used as an action attribute with the name $AimOfRecognition$. Based on this operator schema, the following policy vector should be built, assuming that the start and end time points for this action are equal: $\langle\{\langle ?Student,\ Subject\rangle, \langle ?Mod_1,\ ModuleRecognise\rangle, \langle ?Mod_2,\ ModuleSupport\rangle\},$ $!recognise,\ \{\langle ?Aim,\ AimOfRecognition\ \rangle\},\ \langle\langle ?CurDay,\ ?CurMonth, ?CurYear\rangle,\ \langle ?CurDay,$ $?CurMonth,\ ?CurYear\rangle\rangle\rangle$[9].

5.3.2.3 Methods

Methods define possible decompositions of compound tasks and compound actions into lower level task networks. For each compound task and action, several alternative methods can exist. Methods for compound actions differ from methods for compound tasks, as methods for compound actions additionally contain constructs of operators (effects that specify modifications of the planner's world state and policy parameters tuples that specify parameters for generation of policy requests).

Definition 5.3. Compound action decomposition method $meth_{CA}$ is a construct that defines the compound action execution procedure. A compound action decomposition method is a tuple:

$$\begin{aligned} meth_{CA} = \langle & task(meth_{CA}), duration(meth_{CA}), precondition(meth_{CA}), \\ & policyPar(meth_{CA}), network(meth_{CA}), effect^{+}(meth_{CA}), effect^{-}(meth_{CA})\rangle \end{aligned} \tag{5.14}$$

[9] For operator instances, variables will be substituted with corresponding values.

where $task(meth_{CA})$ is the method's head, the compound action task atom that this method can be applied to, $network(meth_{CA})$ is the task network that this compound action is decomposed into, $precondition(meth_{CA})$ is the precondition expression, $duration(meth_{CA})$ is the duration expression, $policyPar(meth_{CA})$ is the policy parameters tuple, $effect^{+}(meth_{CA})$ and $effect^{-}(meth_{CA})$ are, respectively, the positive and negative effects of the method□

Compound action decomposition methods (as well as compound task decomposition methods) are specified within the planning domain using method schemas that contain variables and constant terms, similarly with operators. A method is relevant for a ground compound action task atom TA_j if there is a substitution for all variables in the method's head such that $task(meth_{CA})$ is equal to TA_j (the same is true for compound task decomposition methods). Compound action decomposition method schemas have one construct that is absent in operator schemas: $network(meth_{CA})$. It is the task network that should be carried out in order to accomplish the compound action. This task network represents the lower level routine for the execution of this compound action. Other compound action decomposition method constructs are defined similarly with the operator's constructs. $duration(meth_{CA})$ is the expression that is evaluated before the execution of the method in order to determine the end time point of the compound action. $precondition(meth_{CA})$ is the precondition expression that determines if this method is applicable to a planner's world state. Preconditions are specified as expressions that can be evaluated to true or false. Method $meth_{CA}$ is *applicable* if its precondition expression is satisfied in the current planner's world state before the execution of the method. $policyPar(meth_{CA})$ is a tuple that contains parameters for the policy request generation. Methods have the same structure of policy parameters tuples and the same definition of legitimacy as operators. Correspondingly, the same structure for the policy vector $polVec(meth_{CA})$ and the same procedure for its generation are used.

Each method schema represents a set of method instances that are derived when its variables are instantiated. It should be noted that values used in the head of a compound action decomposition method should determine uniquely the substitution that is used to satisfy its precondition and the action duration (therefore, they also determine uniquely its effects). In what follows, it is assumed that a method denoted by $meth_{CA}$ (or $meth_{CT}$) is a method instance.

Constructs $effect^{+}(meth_{CA})$ and $effect^{-}(meth_{CA})$ represent effects of a compound action decomposition method. They have the same structure as corresponding constructs for operators. So the execution of compound tasks using decomposition methods can also result in planner's world state updates. This is designated using the function $Apply^{CA} : S \times MethCA \rightarrow S$, where $MethCA$ is the set of all compound action decomposition method instances within the planning domain.

Definition 5.4. Compound task decomposition method $meth_{CT}$ is a construct that defines the compound task execution procedure. It has the same structure as the compound action

decomposition method with the difference that the effects, duration and policy vector constructs are absent□

Applicable (and legitimate) method instances are used during the planning to decompose relevant compound actions and compound tasks in task networks. If method $meth$[10] decomposes compound action or compound task TA_i in a task network TN, task network $network(meth)$ is used to substitute TA_i within the task network TN. All ordering constraints for TA_i defined in TN are applied to all tasks in $network(meth_{CA})$. Considering the nested task network structure, defined in Section 5.3.2.1, it is required only to substitute the decomposed task TA_i in the original task network TN with the new task network. When this done, required ordering constraints are applied automatically. As opposed to compound tasks, during the execution of compound action decomposition methods, additional precondition checks are required. When a compound action TA_c has been decomposed using method $meth_{CA}$ within a task network TN and task network $network(meth_{CA})$ has been successfully executed, the effects of $meth_{CA}$ and time updates should be applied to the current planner's world state. The execution of the task network $network(meth_{CA})$ may result in the planner's world state modifications and current time updates, so the state produced after the $network(meth_{CA})$ execution should be checked. Preconditions of $meth_{CA}$ should be also satisfied in this state, as preconditions guarantee the correct execution of the action's effects. Time value in this state should be equal or less than the new time value that will be applied by $meth_{CA}$[11] (end time point for the compound action is determined using the $duration(meth_{CA})$ expression in the corresponding method schema before the decomposition of the compound action).

5.3.3 Plan representations

The policy-based planner produces conditional plans that, in addition to actions, contain conditions that should be evaluated during the plan execution. These conditions are specified based on information that is available only during the plan execution and is not available during the planning. So these conditions are used to ensure that the execution of the plan will satisfy all requirements specified by the policy authors in their policies even if these requirements cannot be evaluated during the planning.

A plan that is produced by the planning algorithm has a linear form. The planning engine executes actions (using operators) in an order, according to which they will be carried out during the execution of the plan. Therefore, when an action is executed, it is inserted in the plan at the position next to the previous action performed. When a compound action TA_c

[10] Denotes either method $meth_{CA}$ for compound actions or method $meth_{CT}$ for compound tasks.

[11] Actions produced during the execution of $network(meth_{CA})$ should be within the time interval for the action TA_c, as they specify the procedure for its execution on a lower level of detail. The strict equality is not obligatory because compound actions are used as independent actions that have their own preconditions, effects and duration, which are not inferred from the lower level actions. Moreover, the specification of actions on a lower level is optional, meaning that for some compound actions lower level routines can be unspecified.

is decomposed using a method $meth_{CA}$, it is added into the plan using auxiliary primitive actions $!CA_start(TA_c, \langle C_B, C_D, C_A \rangle)$ and $!CA_end(TA_c, \langle C_B, C_D, C_A \rangle)$ that designate the start and end points of the compound action ($\langle C_B, C_D, C_A \rangle$ are condition sets that were generated during the evaluation of policies for this action). Correspondingly, $!CA_start$ is added before $network(meth_{CA})$ is executed by the planning engine. Action $!CA_end$ is added after its execution. Between these auxiliary actions, actions that were carried out by the planning engine during the execution of $network(meth_{CA})$ are placed.

Thus, the plan that is originally produced by the planner is a tuple $Plan^{lin} = \langle a_1, a_2, \ldots, a_n \rangle$, where a_i is an action structure representing a primitive action:

$$a_i = \langle TA_i, \langle C_B, C_D, C_A \rangle \rangle \tag{5.15}$$

where TA_i is a task atom for a primitive actions, C_B, C_D, C_A are the sets of conditions that were generated during the evaluation of policies for this action.

Primitive sctions $!CA_start$ and $!CA_end$ are used in linear plans along with other primitive actions and designate the start and end time points of the corresponding compound actions. In order to get a hierarchical representation of this plan, it should be processed by a converter that transforms a linear plan $Plan^{lin}$ into its hierarchical representation based on the auxiliary actions $!CA_start$, $!CA_end$. A hierarchical plan is a tuple $Plan^{hier} = \langle \mathbf{a}_1, \ldots, \mathbf{a}_z \rangle$, where $\mathbf{a}_m = a_m$ if $\mathbf{a}_m$ represents a primitive action within the a_m action structure or $\mathbf{a}_m = \langle TA_i, Plan_i^{hier}, \langle C_B, C_D, C_A \rangle \rangle$ if it represents a compound action (TA_i is a compound action task atom itself). $Plan_i^{hier}$ is a hierarchical plan that should be executed in order to execute this compound action. C_B, C_D, C_A are the before, during and after conditions that were generated during the evaluation of policies for TA_i. For example, if there is a linear plan $Plan^{lin} = \langle \ldots, a_1, a_2, a_3, a_4, a_5, a_6 \ldots \rangle$, where $a_2 = \langle !CA_start(\&Make_transfer(\ldots), \ldots), \langle C_1, C_2, C_3 \rangle \rangle$, $a_3 = \langle !Pass_assessment(\ldots), \emptyset \rangle$, $a_4 = \langle !Recognise(\ldots), \emptyset \rangle$, $a_5 = \langle !CA_end(\&Make_transfer(\ldots), \ldots), \langle C_1, C_2, C_3 \rangle \rangle$, this plan will be converted into the following nested hierarchical plans structures: $Plan_1^{hier} = \langle \ldots, \mathbf{a}_1, \langle \&Make_transfer(\ldots), Plan_2^{hier}, \langle C_1, C_2, C_3 \rangle \rangle, \mathbf{a}_5 \rangle$ and $Plan_2^{hier} = \langle a_3, a_4 \ldots \rangle$ (see Figure 5.3).

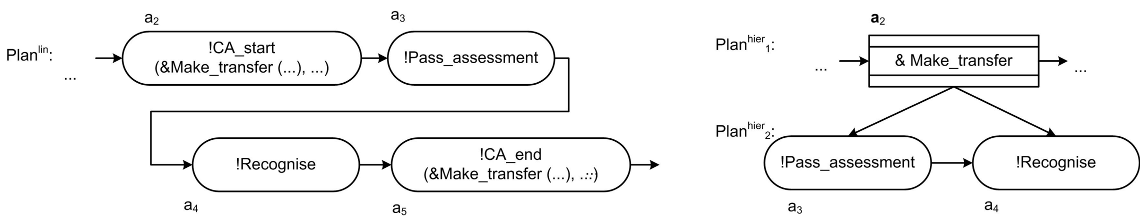

Figure 5.3: Example of hierarchical plan generation

5.3.4 Obligations processing

The obligations returned in a policy decision tuple OblStr during the evaluation of policies for a primitive or compound action (e.g., $PolEval(s, PolVec(o)) = \langle d, \mathit{OblStr}, C\rangle$), are processed by the planning engine. The before-, during- and after-obligation sections Obl_B, Obl_D and Obl_A within the obligations tuple OblStr are converted into task networks. The overall structure of each obligation section Obl_B, Obl_D and Obl_A ($\{\langle Obl_{11}, \ldots, Obl_{1n}\rangle, \ldots, \langle Obl_{m1}, \ldots, Obl_{mk}\rangle\}$) corresponds to the structure of task networks (see Section 5.3.2)[12]. Individual obligations Obl_i used in these sections are represented as task atoms (any task atom types can be used as obligations).

Task networks produced based on before- and after-obligations (Obl_B^{TN}, Obl_A^{TN}) are executed by the planning engine respectively before and after the action that was evaluated. During-obligations can be generated only for compound actions. Task networks produced based on them (Obl_D^{TN}) are used to decompose this compound action, similarly as it is decomposed by a compound action decomposition method. Suppose, during the policy request evaluation for compound action TA_c, before-, during- and after-obligation task networks Obl_B^{TN}, Obl_D^{TN} and Obl_A^{TN} were generated. Additionally, a compound action decomposition method is applied to the compound action TA_c and produces a task network $network(meth_{CA})$. Then, the generated obligation task networks extend the task network $network(meth_{CA})$ according to the following task network structure: $\langle Obl_B^{TN}$, $\{Obl_D^{TN}$, $network(meth_{CA})\}$, $Obl_A^{TN}\rangle$. This new task network guarantees that before- and after-obligations are carried out before and after the task network representing the evaluated action. During-obligations can be executed in any order with the task network $network(meth_{CA})$, produced by the decomposition method. As before-obligations can change the planner's world state when they are executed, after their execution (and before the TA_c execution) the preconditions and policies for TA_c should be re-evaluated. The preconditions should be satisfied and the policy request should be permitted and produce the same before-obligations as were just carried out (if the obligations are not equal, this means that the resulting plan will not satisfy the actual policies).

Policy obligations are used in policy-based planning as the means to extend task networks produced during task decompositions. Using obligations, the same compound tasks or actions being decomposed using the same method can be executed by different task networks. These task networks extend the task network specified in the applied method in order to satisfy specific requirements, which are imposed using policies and which depend on a specific situation when this method is applied to. Examples of task network extensions introduced using obligations are presented in Figure 5.4. The compound action decomposition method $meth_{CA}$ decomposes task $\&Degree$ into three sequential primitive actions: $\langle !Admit, !Study, !Graduate\rangle$. Different extensions of this method using obligations are designated using thick arrows. In the first case, for the

[12]Further, we will use the function $PolEval^TN$ that behaves similarly with $PolEval$, but returns an obligation structure with task networks produced based on the corresponding obligations: $\langle Obl_B^{TN}, Obl_D^{TN}, Obl_A^{TN}\rangle$.

primitive action $!Admit$, some policy requires that an entrance exam should be passed before a student can be admitted (represented using primitive action $!Pass_exam$). The extended task network produced when this obligation is added is shown using dotted lines. In the second case, some policy requires that after the primitive action $!Study$ is executed, a student should pass an assessment (e.g., a government evaluation) before he (or she) can graduate. In the third case, when the compound action $\&Degree$ itself is evaluated, the before-obligation $Preliminary_course$ is returned. This designates that before studying a degree, a student should successfully finish a preliminary course.

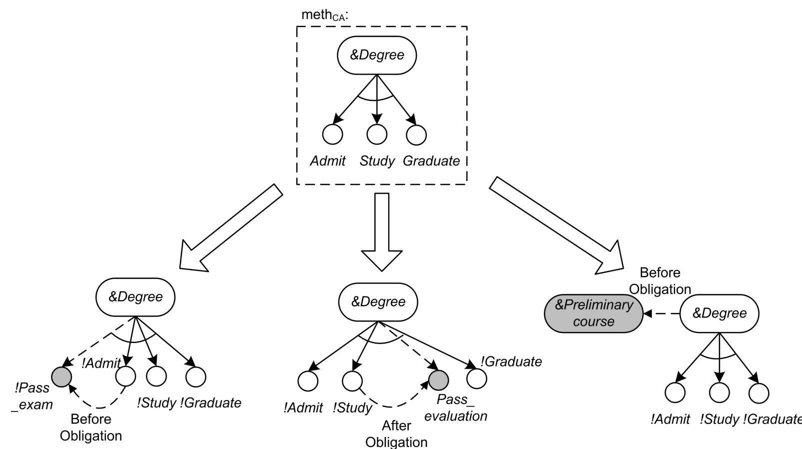

Figure 5.4: Different variants of task execution

5.3.5 Planning algorithm

During the planning, it is required to evaluate policies based on information from the current planner's world state. Hence, the current planner's state should be fully specified during the process of planning in order to have the possibility to retrieve the required information. The utilisation of HTN planning technology where tasks and actions are processed in the same order as they will be carried out during the plan execution provides the means to design a planner where at each planning step the current planner's state is fully specified. This type of HTN planner (see [116, 118]) was taken as a basis for the planning algorithm design and was extended with processing routines for the novel constructs introduced earlier, that is, the compound actions and compound actions decomposition methods, policy conditions and conditional plans, policy obligations and task networks generation based on obligations, initiation of policy evaluation. A novel stack-based mechanism for controlling the overall planning routine and processing of introduced constructs was designed.

A planning problem for the policy-based planner is defined as a tuple $\langle s_0, TN, PlanD\rangle$, where s_0 is an initial state, TN is a task network that is used to specify a task which should be solved, $PlanD$ is a domain description. The planning algorithm that is used to find a solution plan solving

the task network TN from the state s is presented in Figure 5.5. This algorithm produces a linear plan $Plan^{lin}$ that can be executed from the state s, contains only applicable and legitimate actions and is generated from a decomposition tree that was built for the initial task network TN and uses only applicable (and legitimate) methods and operators. A planning algorithm is a recursive function that at each iteration chooses and processes one task from the current task network TN using an operator or a method and makes corresponding updates of the current task network and the planner's world state. Then, it makes a recursive call in order to process the next task from TN. Tasks for processing are chosen in the same order as they will be carried out when the resulting plan is executed.

A task network TN can be partially ordered, hence at each iteration only tasks that can be executed first can be selected. A primitive action TA_i is executed by operator instance o which is applicable and legitimate for the current state s and the action TA_i. When a primitive action is executed, it is removed from the current task network TN. The planner's world state is updated according to the operator's effects. Additionally, when a primitive action is processed, the corresponding action structure $a_i = \langle TA_i, \langle C_B, C_D, C_A \rangle \rangle$ is created where the action itself and conditions generated are saved. This action structure is added into list $Plan$, where the plan is constructed. A compound task is decomposed by an applicable method: a task being decomposed is substituted by a task network specified in $network(methCT)$ part of the method. A compound action is decomposed by an applicable and legitimate method. Auxiliary tasks $!CA_start(TA_i, C)$ and $!CA_end(TA_i, C)$ are added into the task network to mark its start and end points. Auxiliary actions $!CA_start$ and $!CA_end$ are executed using fictitious operators that do not introduce any modifications into the planner's world state. During the execution of $!CA_end$ action, a check is carried out that the current time value do not exceed the time values that were calculated as end time point for the compound action (see Section 5.3.2.3).

A stack-based mechanism was developed for processing of obligations and execution of compound actions. Compound actions and task networks produced based on obligations should be processed using special routines that guarantee that each compound action or obligation task network is executed without interleaving with other tasks[13] (i.e., in a nested manner). Additionally, as was specified in Sections 5.3.4 and 5.3.2.3, after the execution of before-obligations and compound actions, special procedures should be carried out. For example, tasks produced during the decomposition of a before-obligation compound task Obl_i^{TN} should be executed before the action TA that the obligation Obl_i^{TN} has been generated for. When the before-obligation Obl_i^{TN} was executed, the applicability and legitimacy of the original operator that had been applied to action TA should be re-evaluated.

Therefore, during planning, the $Stack$ is used to split the decomposition process into several

[13]When the planning function is called, any action can be chosen if for this action there is no precedence relation.

stages to track the execution of the compound actions and obligations and carry out corresponding checks and activities after their execution. Task networks that are temporarily used as root task networks for the planning algorithm (i.e., it is a task network from which the planner chooses a task for processing at the each stage of the planning process) are saved on to this stack. For example, when Obl_B^{TN} is received during a policy request evaluation, it is saved on to $Stack$ and, further, the planning procedure will be carried out only within the task network Obl_B^{TN}. When all tasks in Obl_B^{TN} are decomposed and executed, Obl_B^{TN} is retrieved from the stack and the operator that has produced this obligation is re-evaluated in the new state. If it is not legitimate or not applicable or new before-obligations are not equal to the previously executed, the planner backtracks. A stack is required since obligations and compound actions can be nested, for example, during the execution of an action within one before-obligations task network Obl_B^{TN}, other before-obligations $Obl_B^{TN'}$ can be produced. When $Obl_B^{TN'}$ is fully decomposed, this task network is removed from the stack and the previous task network on the stack is used as root task network. Compound actions and after-obligations are also placed on the stack when their execution is started. When a compound action is executed, the planner re-evaluates preconditions and applies effects for the corresponding compound action decomposition method.

5.4 Policies

For the specification of policies in the policy-based planner, the XACML policy language is used. This section describes the extensions introduced into the XACML policy language and conventions for the specification of policies that can be used in the policy-based planner and a policy request, which is used to pass information from the planning engine to the policy engine.

5.4.1 Policy request

In the policy-based planner, policy requests are generated for actions and compound actions based on the information specified in their policy vectors: $\langle\{\langle Obj_{ID_1}, Role_1\rangle, \ldots \langle Obj_{ID_n}, Role_n\rangle\}$, $TA^S, \{\langle AParVal_1, AParName_1\rangle, \ldots, \langle AParVal_m, AParName_m\rangle\}, \langle ActBeg, ActEnd\rangle\rangle$. Information about the designated objects $Obj_{ID_1}, \ldots, Obj_{ID_n}$ that can be required during the policy evaluation is stored in a current planner's world state. So during the policy request generation, this information is extracted from the planner's world state and is added into the policy request along with the information contained in the policy vector. Policy requests in the policy-based planner have the following structure:

$$Req = \langle\{DesignObj_1, \ldots, DesignObj_n\}, ActionPar, TimePar\rangle \tag{5.16}$$

where $DesignObj_i$ are constructs representing the designated objects, $ActionPar$ is a construct representing action parameters and $TimePar$ is a construct representing time parameters. This

Global: $Stack$ = $\langle TN \rangle$, $Plan^{lin}$ = $\langle\rangle$.

$PPplan$(state s,domain $PlanD$)

0. $TN = get(first(Stack))$

Case 1: TN is not empty. Nondeterministically choose TA_i such that no other task is constrained to precede TA_i in TN:

1. If TA_i is primitive task:

  1.1 Nondeterministically choose operator o that is relevant for TA_i, applicable and legitimate in s ($PolEval^{TN}(s, polVec(o)) = \langle d, \langle Obl_B^{TN}, , Obl_A^{TN} \rangle, C \rangle$).

  1.2 **If** ($Obl_B^{TN} \neq \emptyset$) **then**

    $push(\langle Obl_B^{TN}, TA_i, o \rangle, Stack)$

    Return $PPplan(s, PlanD)$

   else $ApplyOperator(TA_i, C, Obl_A^{TN}, s, o, PlanD)$ **endif**

2. If TA_i is compound task:

  2.1 Nondeterministically choose method $meth_{CT}$ that is relevant for TA_i and applicable in s.

  2.2 Substitute task TA_i in TN with $network(meth_{CT})$

  2.3 **Return** $PPplan(s, PlanD)$

3. If TA_i is compound action:

  3.1 Nondeterministically choose method $meth_{CA}$ that is relevant for TA_i, applicable and legitimate in s ($PolEval^{TN}(s, polVec(meth_{CA})) = \langle d, \langle Obl_B^{TN}, Obl_D^{TN}, Obl_A^{TN} \rangle, C \rangle$).

  3.2 **If** $Obl_B^{TN} \neq \emptyset$ **then**

    $push(\langle Obl_B^{TN}, TA_i, meth_{CA} \rangle, Stack)$

    Return $PPplan(s, PlanD)$

  Else $ApplyCA(TA_i, C, meth_{CA}, Obl_D^{TN}, Obl_A^{TN}, TN, s, PlanD)$ **endif**

Case 2: TN is empty. Pull value from $Stack$:

1. If $\langle Obl_B^{TN}, TA_i, x \rangle$ was pulled (x is o or $meth_{CA}^{TN}$):

   1.1 **If** x is not applicable or is not legitimate in s **then Return** failure **endif** ($PolEval^{TN}(s, polVec(x)) = \langle d, \langle Obl_B'^{TN}, Obl_D'^{TN}, Obl_A'^{TN} \rangle, C' \rangle$)

   1.3 **If** $Obl_B^{TN} \neq Obl_B'^{TN}$ **then Return** failure **endif**

   1.4 **If** x is an operator **then** $ApplyOperator(TA_i, C', Obl_A'^{TN}, s, x, PlanD)$

    else $ApplyCA(TA_i, C', x, Obl_D'^{TN}, Obl_A'^{TN}, TN, s, PlanD)$ **endif**

2. If $\langle TN_{CA}, methCA, Obl_A^{TN} \rangle$ was pulled:

   2.1 **If** $methCA$ is not applicable in s **then Return** failure **endif**

   2.2 **If** $Obl_A^{TN} \neq \emptyset$ **then**

     $push(\langle Obl_A^{TN} \rangle, Stack)$

     Return $PPplan(Apply^{ca}(s, meth_{CA}), PlanD)$

    Else Return $PPplan(Apply^{ca}(s, meth_{CA}), PlanD)$ **endif**

3. If $\langle Obl_A^{TN} \rangle$ was pulled:

   3.1 **Return** $PPplan(s, PlanD)$

4. If TN was pulled:

   4.1 **Return** true.

Figure 5.5: Planning algorithm

$ApplyOperator(TA_i, C, Obl_A^{TN}, s, o, PlanD)$
1. Add action structure $\langle TA_i, C\rangle$ to tail of plan $Plan^{lin}$
2. Substitute task TA_i in $first(get(Stack))$ with $\emptyset$
3. **If** $Obl_A^{TN} \neq \emptyset$ **then** $push(\langle Obl_A^{TN}\rangle, Stack)$ **endif**
3. **Return** $PPplan(Apply^{op}(s, o), PlanD)$

$ApplyCA(TA_i, C, methCA, Obl_D^{TN}, Obl_A^{TN}, TN, s, PlanD)$
1. $TN_{CA} = \langle !CA_start(TA_i, C), \{network(methCA), Obl_D^{TN}\}, !CA_end(TA_i, C)\rangle$
2. Substitute TA_i in $first(get(Stack))$ with $\emptyset$
3. $push(\langle TN_{CA}, methCA, Obl_A^{TN}\rangle, Stack)$
4. **Return** $PPplan(s, PlanD)$

Figure 5.6: Planning algorithm (cont.)

definition extends the standard XACML policy request definition, which is based on the '*subject-action-resource-environment*' model (Chapter 4). In addition to the subject and resource parts, policy requests in the policy-based planner can contain specifications for several designated objects with different roles. The specifications of these objects are based on constructs utilised in standard XACML requests: named attributes and tree-structured elements containing additional information about the object. So designated objects are specified within the policy request according to the following structure:

$$DesignObj_i = \langle\{\langle\text{'}\mathbf{id}\text{'}, Obj_{ID_i}\rangle, \langle\text{'}\mathbf{type}\text{'}, Obj_{Type_i}\rangle, \langle\text{'}\mathbf{role}\text{'}, Role_i\rangle, \langle ParName_1^i, ParVal_1^i\rangle, \ldots \langle ParName_k^i, ParVal_k^i\rangle, \}, Obj_{Cont_i}\rangle \quad (5.17)$$

The specification of the designated object contains a set of named attributes, represented as 'name-value' pairs, and an object context. The set of named attributes contains several obligatory attributes including the object-term for the designated object Obj_{ID_i} (i.e., its identifier), its type Obj_{Type_i} extracted from the planner's world state (see Section 5.3.1) and its role in the action $Role_i$. Other attributes of the designated object are stored in the planner's world state as binary property-literals, that is, the property-literals that have two terms and one of which is the object-term of the designated object, like $p(Obj_{ID_i}\ \tau^c)$. These literals are retrieved and represented as attributes with attribute names $ParName_j^i = p$ and attribute values $ParVal_j^i = \tau^c$[14]. So during the policy request generation, all binary property-literals for the designated objects are represented in the policy request using named attributes that can be retrieved during the policy evaluation using Attribute Designators based on their names. All other required information about the des-

[14] The order of terms within the literal is not represented in the 'name-value' attribute structure, from which during the policy evaluation information is retrieved using Attribute Designator policy constructs (see Chapter 4). If the order of terms in binary literals with object-terms, representing designated objects, should be taken into account during the policy evaluation, these literals should be retrieved by Attribute Selectors. In this case, they will be represented within object contexts where the order of terms within literals is explicitly specified (see Section 5.6).

ignated objects can be represented in the policy request using object contexts Obj_{Cont_i} and can be retrieved using Attribute Selector policy constructs. Object contexts are special tree structures that can contain all required information about an object stored within the planner's world state, viz., information represented as binary and non-binary property-literals and relation-literals containing its object-term and information about all property and relation literals with object-terms for objects that are related with this object through the chain of one or several relation-literals. Information is added into the object context only if it can be used during the policy evaluation, that is, if it can be retrieved by an Attribute Selector within a policy applicable to the corresponding policy request. Detailed information about the object context structure and the algorithm for its generation is contained in Section 5.6.

$$\begin{aligned} ActionPar = \{&\langle\text{`}\mathbf{action\text{-}id}\text{'}, TA^S\rangle, \\ &\langle AParName_1, AParVal_1\rangle, \ldots \langle AParName_m, AParVal_m\rangle\}, \end{aligned} \tag{5.18}$$

The policy request construct $ActionPar$, representing information about the action itself, contains only named attributes (see Formula 5.18). Each action has one obligatory attribute containing the task symbol of the corresponding task atom TA^S. All other attributes are action parameters that were specified in the policy vector as tuples $\langle AParVal_y, AParName_y\rangle$. Time parameters $TimePar = \langle\langle\text{`}\mathbf{start}\text{'}, dateBeg\rangle, \langle\text{`}\mathbf{end}\text{'}, dateEnd\rangle\rangle$ represent time values when the action starts and finishes. These values are retrieved from the policy vector elements $\langle ActBeg, ActEnd\rangle$ and are also represented as named attributes: for instance, $dateBeg$ is a date of the action start point in the date format supported by XACML.

This definition of policy request extends the standard XACML policy request structure with the possibility to specify several designated objects with different custom roles within the policy request. Using this extension, it is possible to specify policy requests and, correspondingly, policies for actions that cannot be represented using 'subject-resource' schema (or it is difficult), for example, actions that are applied only to one object or that have relationships with several objects (e.g., in transportation problems in planning it is often required to specify for actions 'from' and 'to' destination objects). However, actually, for the specification of each object within a policy request standard XACML constructs are used (named attributes[15] and tree structures). So for the implementation of the policy-based planner, a mapping for these policy requests and policies to standard XACML policy requests and policies was defined, providing the means to re-use the standard XACML policy engine. Using this mapping, in the XACML policy requests several designated objects are specified as sub-parts of one entity. Details about the implementation are given in Chapter 8.

[15]When the XACML policy request is generated, data types for named attributes are determined using pre-defined function (see Chapter 8).

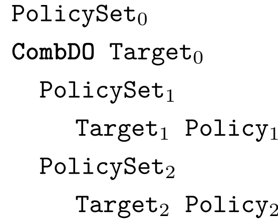

Figure 5.7: Example policy structure

5.4.2 Policy specification

Policies are specified by different authors and contain the specification of constraints that should be enforced during the planning or during the plan execution and additional actions, for example, obligations that should be executed during the planning as ordinary planning tasks or actions. As was described in Chapter 4, different policies and policy sets can be grouped into policy sets for which combining algorithms are specified. These combining algorithms are used to control processing of different policies and resolve conflicts between their decisions when several policies return (different) decisions for the same policy request. For the policy-based planner, the same principle is used.

In order to avoid conflicts between different policies, it is required that the overall policy for the policy-based planner should be specified as one policy set containing other policies nested into this policy set in a hierarchical manner. Within this policy set, different policies and policy sets can be contained. The structure of this policy set and the combining algorithms used in it are not restricted. Using the XACML target mechanism, it is possible to delegate the specification of different policies to different persons and guarantee that these policies are taken into account during the evaluation only in situations for which their authors are responsible. For example, there are two policy authors who can specify policies independently but their policies should be applicable only within their areas of responsibility (e.g., for different actions). Then, the policy structure for *PolicySpec* can be defined (see Figure 5.7), such that a higher-level policy set consists of two sub-policy sets united using the Deny-overrides policy combining operation. When conditions on actions corresponding to each policy author are specified in targets of these policy sets ($[\![\texttt{Target}_1]\!]$ and $[\![\texttt{Target}_2]\!]$), it is guaranteed that these and all lower level policies will be evaluated only when these target conditions are satisfied. Hence, policies $[\![\texttt{Policy}_1]\!]$ and $[\![\texttt{Policy}_2]\!]$, specified by different authors, can be placed into these policy sets and it is guaranteed that they will contribute to a policy decision only within their areas of responsibility.

For the specification of policies referring to designated objects specified according to the extended model of the policy request (see Section 5.4.1), the Attribute Designator element of XACML was extended. In the policy-based planner, this element can refer a named attribute of any designated object contained in the request. The Extended Attribute Designator refers the required

attribute by its identifier and a role of the object within the current action[16]. Similarly, extended target policy elements were introduced that can refer to any designated objects within the policy request. As was described in Section 5.4.1, these policies are converted into an intermediate representation in standard XACML syntax for processing.

5.4.2.1 Conditions specification

Conditions in the policy-based planner are used to specify constraints that should be satisfied at a specific point of the plan execution. Conditions are specified similarly to obligations as part of policies and policy sets (for this purpose, the policy definition in XACML was adapted). They are returned when this policy or policy set returns a Permit decision. Conditions are defined as a tuple:

$$Cond = \langle Position, Constraint \rangle \tag{5.19}$$

where $Position \in \{before, during, after\}$ defines when the condition should be checked relatively to the action for which it is defined. This can be done directly before or after the action, or the condition should hold during the whole action execution interval. *Constraint* defines a condition itself. It can be defined using a natural language, if the constraint will be evaluated by a person, or it can be formally stated as a *Condition* element of XACML, then it can be evaluated by a machine. Conditions returned by different policies are united and are returned by the policy engine to the planning engine for processing.

5.4.2.2 Time constraints

A plan, developed by the policy-based planner, has start and end time points. During the execution of this plan, regulations can be modified and these updates can be planned in advance. So policy authors should have the possibility to specify these future policy updates. The system should know which policies are applicable at a concrete moment in time.

If time constraints are not specified in a policy, this policy applies to all actions. However, some policies are applicable only during specific time intervals which are defined as their operation periods $[Start_{Pol}, Fin_{Pol}]$. All actions carried out by the planning engine should be applicable during all time points between their start and end time points (during the action execution interval). Hence, a policy is applicable to an action if its operation period intersects with the action execution interval: $[Start_{Pol}, Fin_{Pol}] \cap [dateBeg, dateEnd] \neq \emptyset$. Three cases of their intersection are shown in Figure 5.8. In order to implement this condition in a policy, constraints on the action's time interval should be included into its target. They should consider all possible types of intersections: $(Start_{Pol} \leq dateBeg\ and\ Fin_{Pol} \geq dateBeg) or (Start_{Pol} \leq dateEnd\ and\ Fin_{Pol} \geq dateEnd)\ or\ (Start_{Pol} \geq dateBeg\ and\ Fin_{Pol} \leq dateEnd)$.

[16] Attribute Selector policy element can refer to any information within the policy request, so it should not be extended.

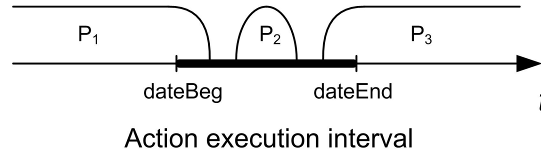

Figure 5.8: Examples of policies, applicable to an action

5.4.2.3 Obligations specification

In policy-based planning, specification of obligations is based on standard XACML obligations with several extensions. Obligations generated by the policy engine are processed by the planning engine: they should be executed as planning tasks. One of the drawbacks of XACML that was detected in Chapter 4 is related to obligations enforcement. In XACML, there are no means to specify a routine how obligations returned along with a policy decision should be executed: as a required order of their execution or as a position relatively to the action that was requested. For the specification of policies in the policy-based planner, this drawback was resolved and corresponding constructs were added for the specification of obligations.

An obligation is specified within a policy or policy set as the following tuple:

$$Obl = \langle Position, TA_i^S, \langle Par_1, \ldots, Par_n \rangle, Order \rangle \tag{5.20}$$

where *Position* is used to specify its position relatively to the action being evaluated, elements TA_i^S and $\langle Par_1, \ldots, Par_n \rangle$ are the task atom symbol and the list of parameters that together specify the compound or primitive action or the compound task that should be executed. *Order* determines the ordering relation between obligations, if several obligations were generated during the evaluation of the policy request.

The position parameter $Position \in \{before, during, after\}$ determines how this obligation should be executed relatively to the action being evaluated. If the position is equal to '*before*' or '*after*', the task, which is specified in this obligation, should be executed directly before or after this action. Position can be equal to '*during*' only if a compound action is being evaluated[17]. So the '*during*' position means that the compound action should be decomposed and the during-obligation is used as task network that is used for the decomposition.

The task symbol TA_i^S in the obligation specification is the task atom symbol that should be used in the task that should be executed. Using the tuple of parameters $\langle Par_1, \ldots, Par_n \rangle$, parameter terms of this task are determined. Each obligation parameter Par_j can contain a term-constant or refer to an attribute within the policy request. In the latter case, $Par_j = \langle AttType, AttrName \rangle$, where *AttType* defines an element within the policy request that the referred attribute belongs to

[17]This is guaranteed by the obligations validation mechanism which will be described in Section 5.4.3

(it can be 'action', 'time' or a role of the designated object), $AttrName$ contains the name of the attribute that should be retrieved. So using the task atom symbol and the parameters tuple, a planning task (compound or primitive action, or compound task) that should be executed by the planner based on the obligation is fully specified.

Several obligations with the same position can be generated for the same policy request, so the planning engine should linearise them. By default, such obligations are processed as an unordered task network. The planning engine will try any possible ordering of tasks representing obligations and will utilise for further planning all task sequences that were successfully executed. However, a policy author may want to enforce a specific ordering for these obligations that the planning engine should follow. For this purpose, into the specifications of obligations an ordering parameter $Order$ should be added that defines the possible order of their execution. The ordering parameter $Order$ can be $\emptyset$ or tuple $\langle Ord_{ID}, Ord_{Num} \rangle$, where Ord_{ID} is an identifier of the ordering relation and Ord_{Num} is a number that specifies the order of the considered obligation within this relation. Obligations with the same ordering relation should be executed sequentially in an ascending order according to this number. When there are several obligations with the same ordering number, they can be executed in any order. Each obligation can refer to only one ordering relation. If an ordering information is not specified for an obligation, it can be executed in any order relatively to other obligations.

As was described in Chapter 4, obligations are specified within different policy sets and policies and are returned from these policies to an upper level of policy evaluation when their policies produce Permit decisions. Obligations in the policy-based planner are produced only for Permit decisions, because only when an action is permitted, it can be executed by the planning engine[18]. The obligations generated during the policy evaluation are enforced in a controlled manner in the policy-based planner using the obligations validation mechanism, which is described in Section 5.4.3.

5.4.3 Obligations validation mechanism

The obligations returned by the policy engine for a policy request have a great impact on the planning process, since they are executed by the planning engine, change the planner's world state and they can be included into the resulting plan. On the other hand, obligation are specified within policies by different policy authors. Using a policy target mechanism, it is possible to specify the scope of policy requests, for which a policy is applicable. However, when a policy becomes applicable, any obligations can be produced (i.e., any obligations that were specified in it by its author). An HTN planning domain should be devised as a set of coordinated methods and an uncontrolled intervention into the planning process is undesirable. So a mechanism is

[18]In a situation when the policy decision is Not applicable, the action is also executed by the planning engine but obligations cannot be returned for this action, since no policies were applicable for it.

required to define the extent to which obligations can intervene into the planning. Additionally, the specification of the constituent policies for a specific policy set can be delegated to different persons, so a person responsible for the whole policy set should have the ability to control obligations being generated by the constituent policies. The standard XACML policy combining algorithms do not provide this ability. In order to satisfy these two requirements, an obligation validation mechanism was proposed.

```
<VALRULE> ::= '(' <COMPACTION> [<BEFOREPART>] [<DURINGPART>] [<AFTERPART>] '=>'
<POLICYLIST> ')' | '(' <PRIMACTION> [<BEFOREPART>] [<AFTERPART>] '=>' <POLICYLIST> ')'
<BEFOREPART> ::= '(' 'before' <ORPART >')' | '*b'
<DURINGPART> ::= '(' 'during' <ORPART> ')' | '*d'
<AFTERPART> ::= '(' 'after' <ORPART> ')' | '*a'
<ORPART> ::= ( <ANDPART> )+
<ANDPART> ::= '(' ['ordered'] <TASK> + ')'
<ACTION> ::= <COMPACTION> | <PRIMACTION>
<COMPACTION> ::= '&' <NAME> | '&' '*'
<PRIMACTION> ::= '!'  <NAME> | '!'  '*'
<TASK> ::= <ACTION> | <NAME>
<POLICYLIST> ::= '(' <POLICY>+ ')' | '*'
<POLICY> ::= <NAME>
<NAME> ::= ('a'-'z'|'A'-'Z'|'0'-'9')+
```

Figure 5.9: Syntax for validation rules

The obligations validation mechanism is based on validation rules. A validation rules registry, $ValidRules$, defines which obligations validation rules are used within which policies: $ValidRules = \{\langle PolicyRef, \{ValRule\}\rangle\}$. The tuple $\langle PolicyRef, \{ValRule\}\rangle$ contains a set of rules that are utilised for policy specified using the policy reference $PolicyRef$. If $PolicyRef$ is empty, these rules are used to validate a whole set of all obligations returned by the policy engine. A grammar for the specification of validation rules is shown in Figure 5.9. Using a validation rule, it is possible to specify which obligations can be generated for a primitive action ($\langle PRIMACTION\rangle$) or a compound action ($\langle COMPACTION\rangle$) being evaluated and which ordering relations should exist for them. A grammar for the validation rules specification is presented in Figure 5.9. Each validation rule has parts for representation of the requirements for before, after and during obligations. Constructs $\langle COMPACTION\rangle$ and $\langle PRIMACTION\rangle$ are used to specify actions for which a rule is applicable. When a set of obligations is produced during the evaluation of a policy request containing task symbol TA^S, these obligations can be validated using rules where constructs $\langle COMPACTION\rangle$ or $\langle PRIMACTION\rangle$ match task symbol TA^S, that is, they are equal or the wild card '$*$' is used in $\langle COMPACTION\rangle$ or $\langle PRIMACTION\rangle$. The obligation rule validates this set of obligations, if before, during and after parts of this rule validate respective sub-sets of this obligation set. Hence, using validation rules it is possible to specify constraints on obligations with different positions and interrelations between them. As defined in the grammar,

for primitive actions only before and after obligations can be used, as it is stated in the obligation definition. Within each validation rule part, several alternative conditions on obligations with the same position are specified. Two types of these conditions can be used: unordered and ordered. An unordered condition requires that all required obligations were generated during the policy evaluation. Using an ordered condition, additionally, it is possible to check that returned obligation can be ordered according to the required order (it should be noted that these obligations can be already partially ordered). Moreover, if an ordered condition is used for the validation, this order is enforced for the validated obligation set. Sub-sets of this obligations set corresponding to rule parts with ordered constraints should be ordered according to the required order. For each action, several validation rules can be specified. For a successful validation, the set of obligations should satisfy one of them. A wild card '$*$' is used to designate the fact that this rule is applicable to any compound or primitive action. A wild card '$*$' with a letter corresponding to the before, after or during validation rule parts designates that this rule permits any set of obligations (even an empty set) in the corresponding position. Optionally, a rule can contain a list of policy and policy set identifiers. If this list is specified, this rule can be used to validate only obligations returned by policies (or policy sets) included in this list.

The validation of obligations, returned during the policy evaluation, can be carried out at different levels. At the top-level, validation rules can be specified by an author of the planning domain. These rules are applied to a result obligation set, returned by the policy engine. An aim of the planning domain author is to guarantee that returned obligations can be executed at this point of the planning based on the modelled environment principles. For example, an entrance exam should be before an admission of student to university. There is no reason to require a student to pass an entrance exam after an admission or during his (or her) studying at university. Policy lists usually are not used at this level, because the planning domain author is responsible for maintaining the validity of all obligations carried out within the environment. Hence, all sets of obligations returned by the policy engine should be validated by a validation rule. When such rule is not found, an Indeterminate decision is produced.

Additionally, validation rules can be specified by policy set authors. These validation rules are used to validate obligations returned by constituent policies for this policy set. Constituent policies generate obligations along with policy decisions. Decisions, returned by these policies, are combined using a combining algorithms. If a result decision matches a decision returned by a constituent policy, the obligations that this policy has generated should be returned by the policy set as its own obligations. When obligations validation rules are specified for this policy set, only obligations validated using these rules can be returned from this policy set. At this level, obligations validation rules can contain policy lists. A policy set author can specify distinct rules for different constituent policies using the lists of applicable policies within the validation

rules. Obligations returned by some policy can be validated only by a policy rule where its policy identifier is specified within the rule's policy list. A validation rule where the wild card symbol '$*$' is specified instead of a policy list can be used for any policy. So at this level, obligations returned by different policies are validated separately. If obligations returned for some policy were not validated using the specified validation rules, the Indeterminate decision is returned.

In Figure 5.10 two examples of validation rules are presented. The first rule defines that an entrance exam obligation can be executed before an admission action. The second rule defines a possible routine for the execution of $transfer_IP$ student transfer action. During the execution of this compound action, a recognition procedure should be carried out (compound action &*Recognise*) and the remaining difference in the EPs should be discarded (action !*Discard_difference*).

```
(!Admit (before (Entrance_exam)) => * )
(&transferIP (during (&Recognise !Discard_difference)) => * )
```

Figure 5.10: Examples of validation rules

5.5 Transformation rules engine

Within the planning environment, the same object properties for different objects can be specified using different terms or units. That is, these terms are used to refer to the same or related notions and the units are used to measure the same characteristic. For example, these terms and units can be adopted in different domains or according to different classification systems. However, specific policies as well as the planner's domain methods and operators are usually specified using a specific set of terms or specific units (i.e., a scale) and cannot interpret other terms and units. The transformation engine operates with rules that specify how to convert values of properties from one scale to another. A basic transformation rule has the following structure:

$$property_2(?OBJ, ?val_2, Scale_2)[, ?val_1 = \langle conversion_expr\rangle] \rightarrow property_1(?OBJ, ?val_1, Scale_1) \tag{5.21}$$

where names of properties $property_1$ and $property_2$ can be equal or different and specify which properties are related (or converted) by this rule. Names of properties correspond to predicate symbols of binary property-literals containing the object-term of the concerned object. $conversion_expr$ is an optional element. It is used to convert numeric values from one scale to another. Otherwise, a mapping from one constant value to another is defined. A property-literal representing the object property that should be converted should be stored in the planner's world state and have the same structure as predicates in the transformation rules.

When the planning or policy engine requires a value of object property in a scale specific to a concrete domain or classification system, it makes a request to the transformation rules engine. In the request, it passes the following values: $\langle property, Obj_{ID}, Scale\rangle$, where $property$ is a name

of the object property that is used as a predicate symbol in the planner's world state. The object that this property describes is represented using its object-term Obj_{ID}. *Scale* is an identifier of the target scale, to which the value should be converted to by the transformation rules engine. During the policy evaluation, these requests are initiated using a special type of attribute designators. In the planning environment, a special function will be implemented that can be used in operators and methods preconditions to request the converted values from the transformation rules engine (see Chapter 8).

When the transformation rules engine receives a request, it analyses its rule base in order to decide if a conversion of the requested property is supported. If it is supported, it processes the request using the literals from the planner's world state, which are used to infer the new value of the property.

5.6 Adaptive object contexts generation technique for policy request construction

As was stated in Section 5.4.1, during the policy requests generation relevant information about designated objects which is stored in the planner's world state and which can be used during the policy evaluation should be extracted and presented as object contexts in the request. An object context is a tree structure that contains required information about the object stored within the planner's world state, viz, information represented as binary and non-binary property-literals and relation-literals containing its object-term and information about all property and relation literals with object-terms for objects that are related with this object through the chain of one or several relation-literals. This section presents a mechanism for selection of the relevant information and its transformation into object context.

This mechanism consists of two sub-techniques. The first technique is a transformation technique that defines how part of the planner's world state that contains information about some object can be transformed into its context. For its implementation, a hyper-graph model of the planner's world state will be introduced. The transformation technique will be defined based on the introduced hyper-graph model of the planner's world state. The second technique is the abstract contexts technique. Using this technique, part of the planner's world state that contains information related to a designated object within a policy request is selected for the transformation into the object context. This object context should be included into the considered policy request. In this technique, policies loaded into the policy repository are pre-processed and abstract object contexts are generated for them. An abstract context specifies what information about an object used as a designated object within a policy request can be used within conditions for these policies. During the policy request generation, based on these abstract contexts concrete contexts for specific designated objects are generated using information stored in the planner's world state.

This technique should select all information that can be used during the policy evaluation, but, on the other hand, it should select only the minimum required information, as this reduces the policy request size.

5.6.1 Hyper-graph of planner's world state and its object model

In Section 5.3.1, the planner's world state was defined as a set of ground positive literals $p(\tau_1^c, \dots, \tau_n^c)$. The following properties are important and should be taken into account during the design of the planner's world state model: the order of terms in a literal is important, one term can be included in one literal several times in different positions, literals with the same predicate symbols can contain different number of terms. The planner's world state can be represented as a hyper-graph $\mathbf{H}_S$. A hyper-graph is a generalisation of a graph where each edge can contain any number of vertices. In $\mathbf{H}_S$, each vertex $t \in \mathbf{T}$ represents an entry of a term in a literal in the planner's world state. Each edge $l \in \mathbf{L}, l \subseteq \mathbf{T}$ is a set of vertices representing term entries related to the same literal.

$$\begin{gathered} \mathbf{H}_S = \langle \mathbf{T}, \mathbf{L} \rangle \\ \mathbf{T} = \{t\}, \mathbf{L} = \{l\} \end{gathered} \tag{5.22}$$

So each reference to the same term in the planner's world state is modelled by different vertices in the hyper-graph $\mathbf{H}_S$. Each edge represents vertices corresponding to terms within one literal. Fictitious literals are not included into the hyper-graph model of the planner's world state, as they are not used during the policy evaluation. All vertices in $\mathbf{H}_S$ are divided into two sets corresponding to object-terms and property-terms: $\mathbf{O} = \{\dots, O_i, \dots\}$ and $\mathbf{P} = \{\dots, P_j, \dots\}$. Similarly, all edges are divided into two disjoint sets corresponding to relation-literals $\mathbf{L}_R$ and property-literals $\mathbf{L}_P$.

$$\begin{gathered} \mathbf{T} = \mathbf{O} \cup \mathbf{P}, \mathbf{O} \cap \mathbf{P} = \emptyset \\ \mathbf{L} = \mathbf{L}_R \cup \mathbf{L}_P, \mathbf{L}_R \cap \mathbf{L}_P = \emptyset \end{gathered} \tag{5.23}$$

The hyper-graph $\mathbf{H}_S$ is labelled using four labelling functions (see Formula 5.24). Edges are labelled using the labelling function η with predicate symbols that are used in corresponding literals. Vertices are labelled according to terms that they represent: object-terms with the labelling function α and property-terms with the labelling function β. Functions α, β and η are not injective, because one term and one predicate symbol can be used in the planner's world state several times. The labelling function ε maps a pair containing one edge and one vertex in the hyper-graph $\mathbf{H}_S$ to an integer number, indicating the sequencing number of the term represented by the vertex within

the literal represented by the edge.

$$\begin{gathered}\alpha : \mathbf{O} \to Term^{Prop}_{OType} \times Term^{Obj} \\ \beta : \mathbf{P} \to Term^{Prop} \ , \ \eta : \mathbf{L} \to Pred \\ \varepsilon : \mathbf{L} \times \mathbf{T} \to \mathbf{N}\end{gathered} \tag{5.24}$$

A hyper-graph $\mathbf{H}_S$ for the planner's world state is not connected (every its edge is included in its own connected component). A connected component is a maximal subgraph where each two vertices are connected by some path. The total number of connected components is equal to $k(\mathbf{H}_S) = |\mathbf{L}|$. An example of hyper-graph $\mathbf{H}_S$ for a planner's world state is represented in Figure 5.11. It contains two objects Obj_1 and Obj_2 with different types, one property for each object and one relation, connecting these two objects. The hyper-graph corresponding to this planner's world state is presented in Figure 5.12. Terms entry labels are shown inside the edges.

a_1 : $\mathbf{Object}(Obj_{ID_1}, Obj_{Type_1})$
a_2 : $\mathbf{Object}(Obj_{ID_1}, Obj_{Type_2})$
a_3 : $Prop_1(Obj_{ID_1}, \tau^c_{Prop1}, \tau^c_{Prop2})$
a_4 : $Prop_2(Obj_{ID_2}, \tau^c_{Prop3})$
a_5 : $Rel_1(Obj_{ID_1}, Obj_{ID_2}, \tau^c_{Prop4})$

Figure 5.11: Example of the planner's world state

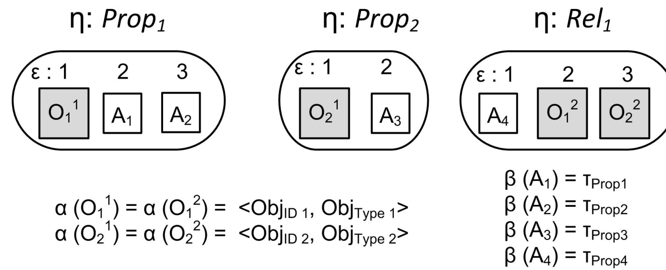

Figure 5.12: Hyper-graph of the planner's world state example

In order to use the hyper-graph model of the planner's world state as a basis for the transformation technique specification, it should represent and distinguish all object-terms, property-terms and literals of the planner's world state and relations between them. Object types are represented in labels. The sequence order of the terms in a literal is represented using terms entry labelling function ε. If one term is included into the same literal several times, it will be represented using different vertices with the same object or property label. Different edges with labels which have the same predicate symbols obviously can include different number of vertices.

In order to retrieve all the required information about an object from the hyper-graph of the planner's world state and represent it in policy request, based on $\mathbf{H}_S$ a planner's world state object model hyper-graph $\mathbf{H}_{Obj}$ is defined. It represents the same information as $\mathbf{H}_S$ with the following

difference. All object vertices with the same object labels are shrunk into one vertex, so edges representing literals with the same object-terms are connected to each other. The object model hyper-graph $\mathbf{H}_{Obj}$ is defined as:

$$\mathbf{H}_{Obj} = \langle \mathbf{O}_O \cup \mathbf{P}_O, \mathbf{L}_O \rangle \tag{5.25}$$

where $\mathbf{P}_O \equiv \mathbf{P}$ and each object-term in the planner's world state is represented by only one vertex is the set $\mathbf{O}_O = \{O_O\}$. So the set of object vertices in the hyper-graph $\mathbf{H}_S$ is divided into equivalence classes. In one equivalence class all vertices are labelled with the same object label $(O_i \sim_\alpha O_j \Leftrightarrow \alpha(O_i) = \alpha(O_j))$. All object vertices $O_i \in \mathbf{O}$ in $\mathbf{H}_S$ from one equivalence class are represented in $\mathbf{H}_{Obj}$ by one object vertex $(O_{O_i} \in \mathbf{O}_O)$ corresponding to this equivalence class $(\mathbf{O}_O \leftrightarrow \mathbf{O}/\sim_\alpha)$. Object labelling function α_O is defined for the set $\mathbf{O}_O$ such that if O_i in $\mathbf{H}_S$ is represented as O_{O_i} in $\mathbf{H}_{Obj}$, then $\alpha_O(O_{O_i}) = \alpha(O_i)$.

The edges of the object model hyper-graph $\mathbf{L}_O$ are defined using the common procedure for vertices being shrunk. Each edge l_j in the hyper-graph $\mathbf{H}_S$ is represented in the hyper-graph $\mathbf{H}_{Obj}$ using edge l_{O_j} such that l_{O_j} is equal to l_j with the difference that object vertices are substituted using the mapping introduced earlier. Edges labelling function η_O and terms entry labelling function ε_O are defined similarly as in $\mathbf{H}_S$. Example of hyper-graph $\mathbf{H}_{Obj}$ for the planner's world state in Figure 5.11 is represented in Figure 5.13.

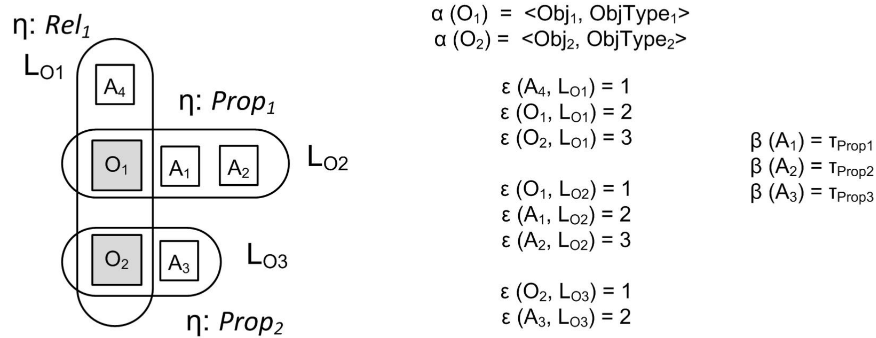

Figure 5.13: Object model hyper-graph for the example planner's world state

The hyper-graph $\mathbf{H}_{Obj}$, obtained using the described rules, is a connected hyper-graph where relations between edges are established using object terms. All vertices representing the same property terms are still different. Correspondingly, the hyper-graph $\mathbf{H}_{Obj}$ is divided into several connected components $k(\mathbf{H}_{Obj})$, each of which represents a set of literals related to each other. When information about a designated object is selected from the planner's world state, only the connected component where its object-term is included should be considered, as the rest part of the planner's world state is not related to this object. The hyper-graph $\mathbf{H}_{Obj}$ has one limitation: a literal that contains the same object-term in different positions cannot be represented in $\mathbf{H}_{Obj}$, as vertices that represent such terms will be shrunk into one vertex. This restriction is avoided at

the level of implementation, because at the level of implementation only hyper-graph model $\mathbf{H}_S$ is stored in full for a planner's world state. When information about relations of object-terms is needed, it is received using a mechanism operating with $\mathbf{H}_S$ and data structures introduced on top of it[19] that returns a set of edges from $\mathbf{H}_S$ with the required object-term.

5.6.2 Abstract contexts

Contexts of designated objects are represented in policy requests as trees. These trees have root nodes corresponding to object-terms of these designated objects. The trees are presented in a request as XML documents. These documents can be accessed during the policy evaluation using *AttributeSelector* elements that are used in policy conditions and contain XPath expressions. Using these XPath expressions, it is specified which information from the object context will be retrieved and, correspondingly, utilised during the policy evaluation. When policies are uploaded into the policy repository, these XPath expressions are analysed and special constructs, called abstract contexts, are created. Abstract contexts are an abstracted and merged representation of all XPath expressions used within different policies. Abstract context AC is specified as a set of abstract context trees AC_{Tree}.

Definition 5.5. Abstract context tree is a tree $AC_{Tree} = \langle V, E, v^R \rangle$. Root vertex v^R represents a set of objects. For a situation when an object from this state is used as a designated object in a policy request, the abstract context tree specifies which literals from the planner's world state related to this object can be requested using *AttributeSelector*s during the policy evaluation (and, hence, which literals should be added into its context). $V = \{\dots, v, \dots\}$ is a set of vertices labelled using function $Func^{OType}$ with object types values ($Func^{OType} : V \rightarrow Term^{Prop}_{OType}$). Special universal vertex v^{ANY} is a vertex for which the label is not specified. Root vertex $v^R \in V$ can be a universal or an ordinary vertex. $E = \{\dots, e, \dots\}$ is a set of edges labelled using function $Func^{Pred}$ with predicate symbols ($Func^{Pred} : E \rightarrow Pred$). Special edge e^{ANY} is used to designate an edge for which the label is not specified□

Each edge in an abstract context tree represents a set of literals in the planner's world state (and, correspondingly, edges in $\mathbf{H}_{Obj}$). Edge $e = \langle v_1, v_2 \rangle$ in AC_{Tree} *matches* edge $l_O = \{t^o_1, \dots, t^o_n\}$ in $\mathbf{H}_{Obj}$ (vertex t^o_x can be an object or property vertex) in relation with vertices t^o_i and t^o_j ($t^o_i, t^o_j \in l_O$) if the following conditions are satisfied: predicate symbol $\eta_O(l_O) = p$ is equal to e's label $Func^{Pred}(e)$ or a universal edge is used; some vertex $t^o_i \in l_O$ matches vertex v_1 and some vertex $t^o_j \in l_O$, such that $t^o_j \neq t^o_i$, matches vertex v_2. When an edge e matches edge l_O in relation to vertices t^o_i and t^o_j, this is designated as $match(e, l_O, t^o_i, t^o_j)$. A universal vertex in AC_{Tree} matches any vertex in $\mathbf{H}_{Obj}$. An ordinary vertex v in AC_{Tree} matches object-vertex O_O in $\mathbf{H}_{Obj}$ if $second(\alpha_O(O_O)) = ObjT(v)$, i.e., their type labels are equal (non-universal vertex in abstract context tree can match only with

[19] For storing information about links between vertices and its fast retrieval, special constructions are defined on top of the $\mathbf{H}_S$ (see Chapter 8).

object vertices in $\mathbf{H}_{Obj}$).

Abstract context trees are built from XPath expressions, contained in policies, and represent location paths used in these expressions. Abstract context trees are constructed in a way that a root vertex corresponds to an object for which a context is built. If in an abstract context tree there is a path from the root to some vertex, in some XPath expression there is a location path that can use during the evaluation of the expression literals matching to edges on this path.

When policies are analysed, several abstract context trees are generated. These trees can contain mergeable paths. A *vertex* v_1 is *mergeable* with vertex v_2 and as a result of merging vertex v_m is produced if $Func^{OType}(v_1) = Func^{OType}(v_2)$, then $Func^{OType}(v_m) = Func^{OType}(v_1)$, or if $v_1 = v^{ANY} \vee v_2 = v^{ANY}$, then $v_m = v^{ANY}$. An *edge* $e_1 = \langle v_{11}, v_{12} \rangle$ is *mergeable* with edge $e_2 = \langle v_{21}, v_{22} \rangle$ into edge $e_m = \langle v_{m1}, v_{m2} \rangle$ if vertices $v_{11} \leftrightarrow v_{21}$ and $v_{12} \leftrightarrow v_{22}$ are mergeable into vertices v_{m1} and v_{m2} and if $Func^{Pred}(e_1) = Func^{Pred}(e_2)$ (then $Func^{Pred}(e_m) = Func^{Pred}(e_1)$) or $e_1 = e^{ANY} \vee e_2 = e^{ANY}$ (then $e_m = e^{ANY}$).

In order to store only distinct abstract trees in an abstract context, when a new abstract tree is generated, it is analysed whether it can be merged with any tree already contained in an abstract context. If a sequence of edges starting from the root vertex or only a root vertex can be merged with the corresponding path of an existing abstract context tree, they are merged[20]. A merged path substitutes corresponding path in an existing abstract context tree and the rest part of the new tree is added to the abstract context tree. If a new tree cannot be merged with any existing tree, it is added as a new one. Using this principle, during the matching of an abstract tree with the planner's world state (or its object hyper-graph), each literal considered at the moment in the planner's state corresponds to only one edge in the abstract context tree.

Using abstract contexts, it is possible to select all literals from the planner's world state that can be utilised during the XPath expressions evaluation within policies. It should be noted that this representation of abstract context trees applies some restrictions on XPath expressions that they can correctly represent. These restrictions are specified in Chapter 8, where a technique for XPath analysis and abstract context trees generation is described.

An overall schema of abstract contexts-based policy requests generation is presented in Figure 5.14. At this schema, two phases are presented separately: the pre-planning phase when the abstract contexts are generated during the policy loading and the planning phase when a policy request is generated with object contexts for its designated objects, based on corresponding abstract contexts. When policies have been loaded, they are analysed and produced abstract contexts are saved into an abstract context registry. In order to separate abstract contexts that are used for specific types of policy requests, a leading variables mechanism is used. It is based on the assump-

[20]If a universal vertex is used as a root in new tree, or some sub-path, starting from the root, consists of universal vertices and edges, all abstract context trees should be merged with these universal vertices and edges.

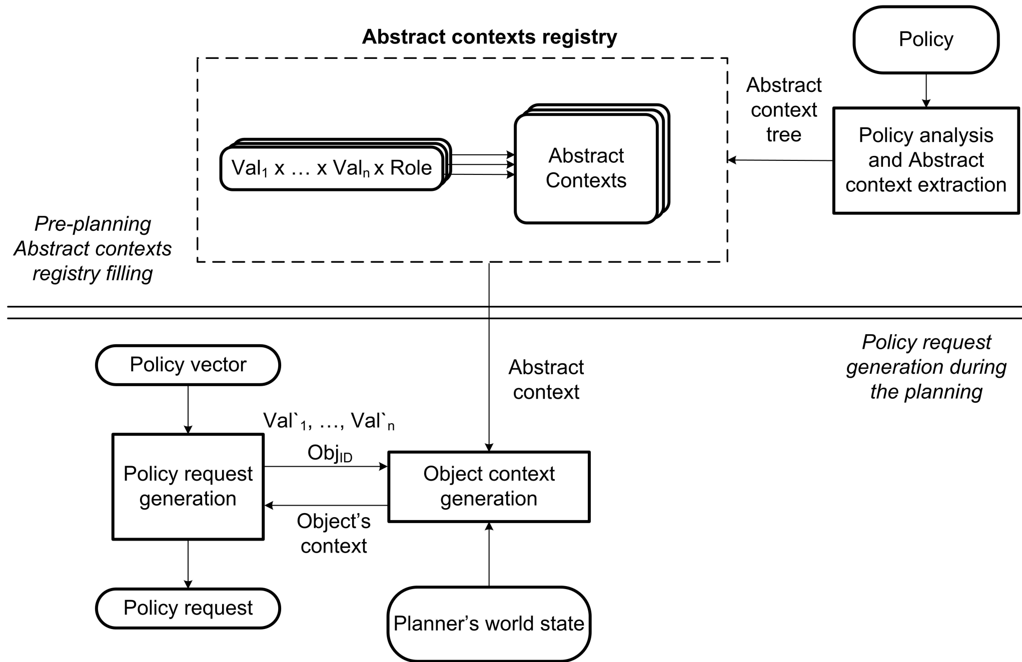

Figure 5.14: Policy request generation schema using abstract contexts

tion that all policy requests and all policies can be divided into classes such that all policies from one class can be applicable only to policy requests from the class corresponding to it. Policy and policy requests can be divided into such classes if in each policy request there is a set of obligatory attributes (called leading variables) and in each policy there is a condition on values of these attributes (or such conditions propagate to a policy from a higher-level policy). The abstract context registry is organised in a way that all abstract contexts referring to one class of policy requests are stored separately. For each combination of leading variables values $Val_1 \times \cdots \times Val_n$ and a designated object's role, distinct abstract context $AC \equiv \{\ldots AC_{Tree} \ldots\}$ is created. So when a policy request is generated, based on current values of leading variables $Val'_1 \times \cdots \times Val'_n$ and a role of the designated object Obj_{ID}, it is possible to determine which abstract context should be used for the concrete contexts generation for the designated objects. Using this mechanism, it is possible to reduce an area of applicability of abstract contexts and, hence, reduce the amount of information that should be extracted from the planner's world state.

5.6.3 Generation of object contexts

A concrete object context, built using corresponding abstract context, represents information from the planner's world state that can be used during the policy evaluation. An algorithm for generation of a concrete context for an object represented as vertex o within $\mathbf{H}_{Obj}$ is presented in Figure 5.15. As an input information, it receives the object o, for which a context should be built, a connected component $k(\mathbf{H}_{Obj})$ containing o and an abstract context AC selected from the registry based on

the policy request. First, an abstract context tree in AC containing the root vertex that matches o is determined. According to the rules of the abstract contexts construction, there is at most one such tree within the abstract context AC. As a source of information for the object context construction, the connected component of the hyper-graph $k(\mathbf{H}_{Obj})$ is used. The abstract context tree is used to determine which edges from it should be converted and represented as the object context. During the object context construction, these two graphs are analysed and a third graph, the resulting object context tree, is created.

```
Generate(Object vertex o, Hyper-graph k(H_Obj), AbsCont AC)
1.  Select AC_Tree in AC, such that o matches v^R (v^R ∈ AC_Tree)
2.  If there is no such AC_Tree then Return empty context ⟨⟩ endif
3.  CurV_AC := v^R; CurV_Hobj := o
4.  Add CurV_Cont := clone(o) into ContextTree
5.  Call Analyse(CurV_AC, CurV_Hobj, CurV_Cont, ContextTree)
6.  Return ContextTree.
Analyse (vertex CurV_AC,vertex CurV_Hobj,vertex CurV_Cont, context ContextTree)
1.  Loop 1 for each edge l_O adjoining with CurV_Hobj in k(H_Obj)
2.  Loop 2 for each vertex t^o ∈ l_O not equal to CurV_Hobj
    2.1.  If there is child v for CurV_AC in AC_Tree (e = ⟨CurV_AC, v⟩) such that
match(e, l_O, CurV_Hobj, t^o) then:
        2.1.1.  If l_O was not added into ContextTree during current loop 2 cycle
        then:
          2.1.1.1.S := clone(l_O), convert S into Bipartite graph G_K(S)
          2.1.1.2.Add G_K(S) into ContextTree by merging vertex CurV'_Hobj in
G_K(S) with CurV_Cont (CurV'_Hobj is vertex in G_K(S) cloned from CurV_Hobj)
        endif
        2.1.2.  Analyse(v, t^o, t^o', ContextTree) (t^o' is vertex in ContextTree cloned
from t^o)
    endif
End loop 2
End loop 1
```

Figure 5.15: Object context generation algorithm

At each step of the algorithm, three vertices are used as current vertices within the three graphs being used. At the beginning, the current vertex for the AC_{Tree}, $CurV_{AC}$, is set up with its root and the current vertex $CurV_{HObj}$ for the $k(\mathbf{H}_{Obj})$ - with o. Next, a depth-first search of paths in $k(\mathbf{H}_{Obj})$, matching edges in AC_{Tree}, is started. All vertices in edges of $k(\mathbf{H}_{Obj})$ that are incident with $CurV_{HObj}$ are analysed. If in the AC_{Tree} there is an edge incident with the current vertex $CurV_{AC}$ that can match with an edge of $k(\mathbf{H}_{Obj})$, the whole edge of $k(\mathbf{H}_{Obj})$ is added into the resulting context *ContextTree*. In order to construct *ContextTree* as a tree, this edge is converted into an equivalent Bipartite graph, which represents the same information but presented as an unordered graph. This graph is added into *ContextTree* and is connected with the current vertex $CurV_{Cont}$. When the matching edge is found and its processing is finished, the current vertices in all three graphs are updated and a new iteration of the search process in initiated. As

it can be seen, in this process AC_{Tree} guides the search process. It is used to determine which edges should be added into the context. Moreover, as AC_{Tree} is a tree, cycles, which can exist in $k(\mathbf{H}_{Obj})$, are resolved. When we add an edge to *ContextTree*, we clone it and connect only to one vertex in the *ContextTree* (i.e., the current vertex).

Definition 5.6. Object context *ContextTree* built for object vertex o in $\mathbf{H}_{Obj}$ is a tree $\langle V^C, E^C, v^{R_o} \rangle$, where $V^C = \{\dots, v^C, \dots\}$ is a set of vertices labelled using function $Func^{Hobj}$. Function $Func^{Hobj}$ maps a vertex in the context to an element of the object model hyper-graph $\mathbf{H}_{Obj}$, that is, its vertex or edge: $Func^{Hobj} : V^C \to \mathbf{O}_O \cup \mathbf{P}_O \cup \mathbf{L}_O$. Root vertex is labelled with o: $Func^{Hobj}(v^{R_o}) = o$. Vertices of the context tree represent literals related with the object o[21] in the planner's world state (and, correspondingly, edges of the object model hyper-graph $\mathbf{H}_{Obj}$) that can be used during the policy evaluation. An edge l_{O_j} of the object model hyper-graph $\mathbf{H}_{Obj}$, containing vertices $t^O_1, \dots, t^O_k$, is represented in *ContextTree* as a vertex labelled with l_{O_j} and vertices $v^C_1, \dots, v^C_k$ adjacent with this vertex and labelled with $t_{O_1}, \dots, t_{O_k}$. A vertex v^C_i in *ContextTree* labelled with an object-vertex O_{O_i} has a child vertex labelled with an edge l_{O_m} only if the object-vertex O_{O_i} is included in the edge l_{O_m} in $\mathbf{H}_{Obj}$ □

An example showing how a Bipartite graph for $k(\mathbf{H}_{Obj})$ is transformed into an object context tree based on the abstract context tree is represented in Figure 5.16. At the graphical representation of the context tree, labels produced by a terms entry labelling function $\varepsilon' : V^C \to \mathbf{N}$ are presented. This function is an updated version of the terms entry labelling function ε, which defines ordering of terms in literals in the object model hyper-graph. For a vertex in *ContextTree* representing a term (i.e., labelled with a vertex in $\mathbf{H}_{Obj}$), function ε' returns its sequence number in the literal represented as its parent node. For a vertex in *ContextTree* representing a literal (i.e., labelled with an edge in $\mathbf{H}_{Obj}$), function ε' returns a sequence number in this literal for the term represented by its parent vertex.

The resulting *ContextTree* is converted into an XML document and inserted into the policy request as a designated object context. Each vertex in the context tree is represented as an element. So these elements are nested according to their positions in the tree. Vertices representing object-terms are represented as elements with object types used as tag names. Object identifiers and their sequence numbers in literals (ε') are represented as their attributes. Vertices representing literals are transformed into elements with predicate symbols used as tag names. Vertices representing property-terms are transformed into elements with their sequence numbers in literals used as tag names. Property-terms themselves are stored in the content of these elements.

[21] Object o is represented in the planner's world state by the corresponding object-term.

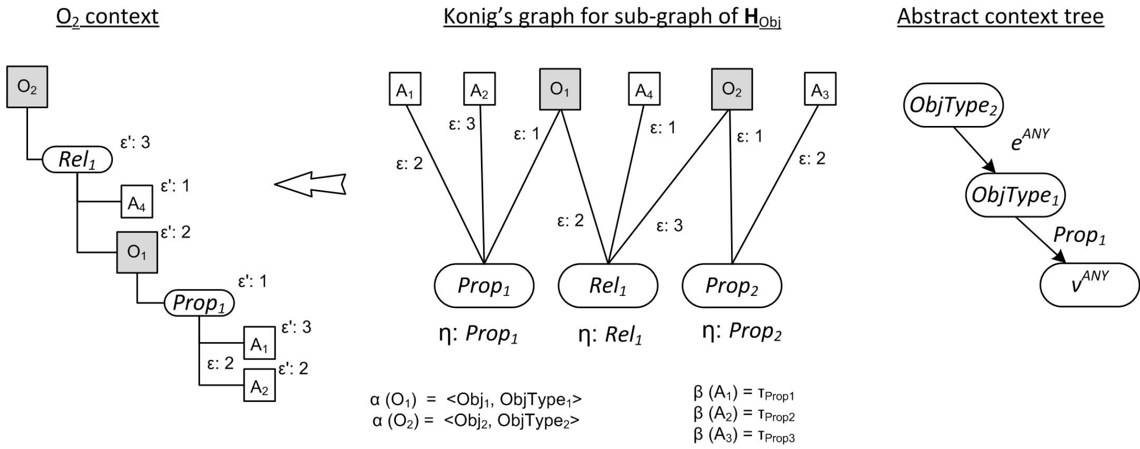

Figure 5.16: Transformation of Bipartite graph for $k(\mathbf{H}_{Obj})$ into an object context

5.7 Conclusion

In this chapter, the policy-based planner was described, which carries out the role of the main processing engine in the CEP generation framework (see Chapter 3). An overview of the policy-based planner, its main components and interaction processes between these components, which form the basis for the policy-based planner operation, was given. The policy-based planner consists of three main components: a planning engine, a policy engine and a transformation rules engine. Each component was described in detail including the specifications that are utilised by each of these components in order to carry out their functions. In the policy-based planner, these three components were integrated into the single planning engine, what provided the means to jointly exploit their advantages during the planning.

The main contribution made in this chapter is a design of the policy-based planner that provides the means to carry out planning in environments with heterogeneous regulations. The policy-based management approach was applied to the new area, the HTN planning technology, extending its possibilities. As was shown in Chapter 2, existing planning technologies do not satisfy requirements for planning in environments with heterogeneous regulations that are different in different domains, manage different aspects of the plans being developed and are specified by different persons independently. Additionally, during the design of the policy-based planner, the following contributions were made. Using the transformation rules, the policy-based planner can operate with different terms and units adopted to designate the same or similar notions and measure the same characteristics. The XACML policy language was extended. The policy specification and evaluation mechanisms were extended in order to have the possibility to specify how obligations produced during the evaluation of a policy request should be executed relatively to each other and to the action being evaluated. The obligations validation mechanism was introduced in order

to have control over obligations produced during the policy evaluation: which obligations can be generated in a current situation,; how they should be executed relatively to each other and to the action being evaluated; which obligations can be produced by a specific policy. Moreover, the adaptive technique for the construction of policy requests, specifically, their object context parts, was designed. The object context represents information about an object stored in the planner's world state. While XACML does not provide mechanisms to determine what information will be required during the policy evaluation, the adaptive object context construction technique was designed to develop object contexts containing only information that can be required during the policy evaluation.

In the planner described in this chapter, policies can be evaluated for an action when all information about the action is fully known. In the next chapter, an extension to the policy-based planner, a postponed policy enforcement mechanism, providing the possibility to evaluate policies at earlier stages of the planning, when not all information about the action is known, is introduced.

Chapter 6

Postponed policy enforcement mechanism

Objectives:

- *Introduce a mechanism that enables the evaluation of policies at earlier stages of the policy-based planning, when not all required information could be available.*
- *Extend the XACML policy evaluation algorithm in order to process policy requests, containing only partial information about the planner's world state.*

6.1 Introduction

Policies in policy-based planning, being specified externally from the planning domain, determine how planning should be carried out in specific situations, that is, which actions should be executed and which actions should be avoided. In the designed policy-based planner, policy requests are generated and evaluated for primitive and compound actions at the moment of their execution. Thus, the information specified using policies is consulted only at the later stage of the planning, that is, at the moment when a specific policy decision inferred based on this information must be enforced during the planning. As was examined within the planning community (e.g., in [157, 34]), the ability of the planner to recognise important events, which should occur during the planning, earlier is critical for the good performance of the planner. Information about the anticipated events and their effects can be used for reasoning at the current stage of the planning. For example, based on this information, conflicts and interactions within the current plan can be detected earlier, so they can be resolved or considered during the choice of future planning actions. Correspondingly, this can help to prune the search space and avoid future backtracking.

This chapter introduces a postponed policy enforcement mechanism providing the means to evaluate policies at earlier stages of the planning. Using it, policy requests can be generated and evaluated for planning actions that should be executed later on during the planning, but for which some information is already available. These are primitive and compound actions that are

contained in the current task network and actions that will be added in it during the planning as a result of compound tasks decomposition. When these policy requests are constructed, not all required information could be available. So new constructs were added into the planning domain in order to differentiate which information can be changed into the planner's world state in the future course of the planning and determine which new information can be added (e.g., which effects can be introduced during the execution of a compound task). For this purpose, high-level effects and increasing/decreasing effects set mechanisms were introduced. Correspondingly, using these mechanisms, partial policy requests can be generated that contain only part of the information that can be required during the policy evaluation. The XACML policy evaluation procedure was extended in order to evaluate such policy requests. When partial policy requests are evaluated, the resulting decision can be indeterminate if the policy request does not contain all the required information. In order to indicate this situation, a new Indeterminate temporal decision was introduced into the XACML policy language. When it occurs, the partial policy request should be postponed and re-evaluated when more information is available to refine it. But the main performance gains of the postponed policy enforcement should be made when an exact decision is produced during the partial policy evaluation. If this decision is Deny, a dead-end is detected and a large part of the search space can be pruned. If this decision is Permit, the evaluation of future policy requests that refine the current request can be eliminated. This leads to the planning time reduction and provides the means to produce the solution faster.

This chapter is organised as follows. First, a number of placeholders are introduced for the specification of partially known objects in Section 6.2.1. These placeholders are used in partial policy requests and constructs designating future modifications of the planner's world states as described in Section 6.2.2. A routine enabling the introduction of partial policy requests during the planning, which guarantees that each partial policy requests represents at least one policy request being carried out later on during the planning, is presented in Section 6.2.3. The overall schema of postponed policy enforcement is described in Section 6.2.4. Later in this chapter, a modified version of the XACML policy evaluation procedure supporting evaluation of partial policy requests is described (see Section 6.3). The partial policy evaluation procedure for XACML is introduced as an extension of the XACML policy evaluation formal model described in Chapter 4. As the partial policy evaluation procedure is defined using the corresponding formal model, it is possible to guarantee that the partial policy evaluation is an extension of the standard XACML policy evaluation procedure and it possesses the required properties (i.e., the monotonicity, see Section 6.3).

6.2 Postponed policy enforcement

6.2.1 Constructs for partially known information specification

The postponed policy enforcement mechanism is based on the possibility to evaluate policies earlier during the planning. During this evaluation, the policy request vector that should be evaluated and the corresponding planner's world state can be known only partially. So for their specification three types of placeholders were introduced. Since during the course of the planning more information becomes known, these placeholders should be substituted by ordinary terms eventually.

Definition 6.1. Null property is a special property-term 'NIL' that can be used as a placeholder for an object's property value. When a 'NIL' term is used in a property- or relation-literal, this means that this literal can be substituted by one or several literals where the 'NIL' term is substituted by different property-terms[1] $\square$

Definition 6.2. Dummy object is an object that is known partially. Its known properties and relations are saved in a special set being supported by the planner (it will be referred as Dummy Objects Space $DumObjSps$). These properties and relations can contain only invariant literals[2]. All other properties and relations of dummy objects are considered as unknown at the current stage of the planning$\square$

Definition 6.3. Hierarchy of properties is a tree $\mathcal{G} = \langle Term^{\mathcal{G}}, E^{\mathcal{G}}, \tau_0^{\mathcal{G}} \rangle$, where $Term^{\mathcal{G}} = \{\dots \tau_r^{\mathcal{G}} \dots\}$ is a set of hierarchical property-terms $\tau_r^{\mathcal{G}} \in Term^{Prop}$, $E^{\mathcal{G}} = \{\dots \langle \tau_l^{\mathcal{G}}, \tau_m^{\mathcal{G}} \rangle \dots\}$ is a set of edges that define the tree structure and $\tau_0^{\mathcal{G}} \in Term^{\mathcal{G}}$ is a root of the tree. All property-terms in $Term^{\mathcal{G}}$ except the leaf terms are used as placeholders: hierarchical property term $\tau_r^{\mathcal{G}} \in Term^{\mathcal{G}}$ can be substituted by any property-term $\tau_k^{\mathcal{G}} \in Term^{\mathcal{G}}$, if $\tau_k^{\mathcal{G}}$ is a descendant for $\tau_r^{\mathcal{G}}$. If in a planning domain there are several hierarchies of properties $\mathcal{G}_1, \dots, \mathcal{G}_f$, their sets of hierarchical terms $Term_1^{\mathcal{G}}, \dots, Term_f^{\mathcal{G}}$ should not overlap$\square$

Dummy objects are specified within the planner's world state similarly with ordinary objects. For this purpose, special predicate symbol **Dummy** is used (see Formula 6.1). Each dummy object is represented as an object-term Dum_{ID}, used for its identification, and has a type Obj_{Type}. In contrast to null properties, one dummy object can be substituted only by one ordinary object. When an object-term Dum_{ID} occurs several times, the same Obj_{ID} should be used for its substitution. In order to store which object-terms Obj_{ID} have been used to substitute specific dummy objects Dum_{ID} and do not substitute them with other object-terms in the future course of the

[1] If several 'NIL' terms are used within one literal, the overall set of terms that substitute the 'NIL' terms should be distinct.

[2] Invariant literals are literals that cannot be modified in the planner's world state, i.e., there are no positive or negative effects within the planning domain specification that have the same predicate symbols.

planning, a substitution $\theta = \{\dots \langle Dum_{ID}, Obj_{ID}\rangle \dots\}$ is used[3].

$$\textbf{Dummy}(Dum_{ID}\ Obj_{Type}) \tag{6.1}$$

Set $Term^{DumObj}$ contains all object-terms representing dummy objects. A set of known properties and relations for a dummy object, contained in the Dummy Objects Space, is returned using the function $DumDescr(Dum_{ID})$.

Definition 6.4. Partially specified literal L_{part} is a literal that contains a dummy object, or a placeholder "NIL", or a non-leaf hierarchical property-term:

$$L_{part} = p(\tau_1^c, \tau_2^c ..., \tau_y^c) \tag{6.2}$$

where p is a predicate symbol and $\exists \tau^c_{d,\ d\in[1,y]}$ $(\tau_d^c = NIL \vee \tau_d^c \in Term^{DumObj} \vee (\tau_d^c \in Term^{\mathcal{G}} \wedge \overline{\exists}\tau_s^{\mathcal{G}}(\langle \tau_s^{\mathcal{G}}, \tau_d^c\rangle \in E^{\mathcal{G}})))\square$

In the following, when a term "literal" is referred, it can be a partially specified literal or a fully known one. Such literals will be denoted as $\tilde{L}$. During the planning, partial literals are refined into more specific literals where some placeholders were resolved and, finally, into a fully known literals. If a literal $\tilde{L}_j = p_j(\tau_{1j}^c, \dots, \tau_{nj}^c)$ is a refinement of literal $\tilde{L}_i = p_i(\tau_{1i}^c, \dots, \tau_{ni}^c)$ considering previous substitutions for dummy objects θ or they are equal, they are connected by a **refinement or equal relation under substitution** θ: $RefEql^l(\tilde{L}_j, \tilde{L}_i, \theta)$[4]. It holds in the following cases:

- Literals are equal: $p_i = p_j$ and $\forall k \in \{1, 2, \dots, n\}$ $(\ \tau_{kj}^c = \tau_{ki}^c\ )$.
- If $p_i = p_j$ and for all $h \in \{1, 2, \dots, n\}$ such that $\tau_{hi}^c \neq \tau_{hj}^c$ one of the following conditions should hold:
  - Term τ_{hj}^c substitutes null-term τ_{hi}^c: $\tau_{hi}^c = NIL$ and $\tau_{hj}^c \in Term_{Prop}$ (designated as $ref_null(\tau_{hj}^c, \tau_{hi}^c)$).
  - Property-terms τ_{hj}^c and τ_{hi}^c are in the same hierarchy of properties $\mathcal{G}$ and τ_{hj}^c is a descendant of term τ_{hi}^c (designated as ref_hier $(\tau_{hj}^c,\ \tau_{hi}^c)$).
  - A substitution of dummy object-term τ_{hi}^c into object-term τ_{hj}^c exists in θ: $\exists \langle Dum_{ID}, Obj_{ID}\rangle \in \theta$ $(\ \tau_{hi}^c = Dum_{ID} \wedge \tau_{hj}^c = Obj_{ID}\ )$.
  - Dummy object-term τ_{hi} is substituted by object-term τ_{hj} under the substitution θ (designated as ref_dum $(\tau_{hj},\ \tau_{hi},\ \theta)$[5]): $\tau_{hi} \in Term^{ObjDum}$ and $\tau_{hj} \in Term^{Obj}$ and $Type(\tau_{hi}) = Type(\tau_{hj})$ and $\forall \tilde{L}_d \in DumDescr(\tau_{hi})$ $(\ \exists L_j \in CurState\ (\ RefEql^l(L_j,\ \tilde{L}_d,\ \theta')\ ))$, where $CurState \in S$ - current planner's world state and $\theta' = \theta \cup \langle \tau_{hi}, \tau_{hj}\rangle$. An object-term τ_{hj} can substitute a dummy object-term τ_{hi} if they have the same type

[3] For each dummy object-term Dum_{ID} only one substitution can be defined.

[4] If the substitution θ is empty, this relation can be designated as $RefEql^l(\tilde{L}_j, \tilde{L}_i)$.

[5] If the substitution is empty, this relation can be designated as $ref_dum(\tau_{hj}, \tau_{hi})$.

and for each known property or relation of τ_{hi} the object τ_{hj} has a property or relation that refines it. When the object τ_{hj} has substituted the dummy-object τ_{hi}, substitution θ is extended with the pair $\langle \tau_{hi}, \tau_{hj} \rangle$.

Additionally, if in $\tilde{L}_i$ the same dummy object-terms are used in different positions, these dummy object-terms should be substituted by the same object-terms in $\tilde{L}_j$.

6.2.2 Partial policy requests

Partial policy requests represent known information about policy requests for actions that will be executed later during the planning. Partial policy requests are generated based on partial policy vectors. They are special policy vectors referring to actions that will be executed during the future course of the planning and, possibly, containing placeholders. Partial policy vectors are attached to tasks in a current task network: to primitive actions, representing that the partial policy request will be generated for this action, and to compound tasks and actions, representing that it will be generated for this action or for an action that will be executed during its decomposition.

In order to introduce a definition for the partial policy vector a loose time interval construct should be introduced. A loose time interval is a time interval $TInterval^p = \langle ActBeg', ActEnd' \rangle$ that is used as a placeholder for a policy request's time interval. It shows that the exact time interval for this request $TInterval$ ($TInterval = \langle ActBeg, ActEnd \rangle$) is unknown but it will be within the specified loose time interval: $[ActBeg, ActEnd] \subseteq [ActBeg', ActEnd']$.

Definition 6.5. Partial policy vector $PolVec^p$ is a policy request vector where dummy objects can be used as designated objects, 'NIL' properties and non-leaf hierarchical properties can be used as action parameters, a loose time interval $TInterval^p$ can be used instead of an exact time interval□

Similarly to partially specified literals, a refinement or equal relation is defined for partial policy vectors. When a partial policy vector $PolVec^p_j = \langle \{ \langle Obj_{ID_1j},\ Role_{1j} \rangle,\ \ldots \langle Obj_{ID_nj},\ Role_{nj} \rangle \}$, $TA^S_j, \{ \langle AParVal_{1j},\ AParName_{1j} \rangle,\ \ldots, \langle AParVal_{mj},\ AParName_{mj} \rangle \},\ \langle ActBeg_j,\ ActEnd_j \rangle \rangle$ is a refinement of partial policy vector $PolVec^p_i$ or equal to it under substitution θ, they are connected by a **refinement or equal relation** $RefEql^r(PolVec^p_j, PolVec^p_i, \theta)$[6]. It holds in the following cases:

- Partial policy vectors are equal: $\forall k \in \{1, \ldots, n\}$ $(\ Obj_{ID_ki} = Obj_{ID_kj} \wedge Role_{ki} = Role_{kj})$ and $TA^S_i = TA^S_j$ and $\forall h \in \{1, \ldots, m\}$ $(\ AParVal_{hi} = AParVal_{hj} \wedge AParName_{hi} = AParName_{hj}\ )$.
- Partial policy vectors are not equal, $TA^S_i = TA^S_j$ and:
  - For all $q \in \{1, \ldots, n\}$ such that $(Role_{qi} = Role_{qj} \wedge Obj_{ID_qi} \neq Obj_{ID_qj})$, $ref_dum(Obj_{ID_qj}, Obj_{ID_qi}, \theta)$.

[6]If the substitution is empty, this relation can be designated as $RefEql^r(PolVec^p_j, PolVec^p_i)$.

- For all $q \in \{1, \ldots, m\}$ such that $(AParName_{qi} = AParName_{qj} \wedge AParVal_{qi} \neq AParVal_{qj} \wedge AParVal_{qi} \in Term_{DumObj})$, $ref_dum(AParVal_{qj}, AParVal_{qi}, \theta)$.
- For all $q \in \{1, \ldots, m\}$ such that $(AParName_{qi} = AParName_{qj} \wedge AParVal_{qi} \neq AParVal_{qj} \wedge AParVal_{qi} = NIL)$, $ref_null(AParVal_{qj}, AParVal_{qi})$.
- For all $q \in \{1, \ldots, m\}$ such that $(AParName_{qi} = AParName_{qj} \wedge AParVal_{qi} \neq AParVal_{qj} \wedge AParVal_{qi}, AParVal_{qj} \in Term^{\mathcal{G}})$, $ref_hier(AParVal_{qj}, AParVal_{qi})$.
- If $TInterval^p_i \neq TInterval^p_j$, then $TInterval^p_j \subset TInterval^p_i$.

If a partial policy vector is attached to task TA, this task should correctly implement this partial policy vector. The task TA correctly implements the partial policy vector $PolVec^p$, if in every possible execution of TA a policy request is generated based on the policy vector $PolVec$ that refines $PolVec^p$ or is equal to it: $RefEql^r(PolVec, PolVec^p)$. A set of partial policy vectors assigned to a task TA is returned by function $requests(TA)$. This property can be specified for each type of the planning tasks as follows:

- A primitive action TA_j correctly implements a partial policy vector $PolVec^p_i$, if the operator that can be applied to TA_j contains policy vector $PolVec_i$ such that $RefEql^r(PolVec_i, PolVec^p_j)$.
- A compound task or action TA_k correctly implements a partial policy vector $PolVec^p_i$, if during each possible decomposition of TA_k an operator or a compound action decomposition method is carried out that contains a policy vector $PolVec_i$ such that $RefEql^r(PolVec_i, PolVec^p_i)$.

When a task is decomposed during the planning, new tasks produced by the decomposition method are assigned with partial policy vectors that were specified for them by the method author. In order to guarantee that all partial policy vectors are correctly implemented by corresponding tasks, the required correctness checks were introduced into operators and methods execution routines (see Section 6.2.3). It should be noted that as partial policy requests are preliminary policy checks, they are specified by the methods' authors optionally[7]. If some partial policy vectors that could be added for tasks produced as a result of the decomposition are omitted, the corresponding policy checks will be done during the execution of primitive and compound actions using the ordinary evaluation mechanism of the policy-based planner.

6.2.2.1 Constructs for future planner's states projection

For the construction of partial policy requests that refer to future planner's world states, partial specifications of these future states should be derived based on the current planner's world state. In order to develop this specification, updates of the current planner's world state that will be

[7]Provided that checks that guarantee correct implementation of partial policy vectors are satisfied.

carried out and updates that might be carried out should be known. For this aim, Increasing and Decreasing effects sets and high-level effects constructs will be defined in this section.

High-level effects

The performance gains of the postponed policy enforcement depend on the possibility to derive exact policy decisions earlier during the planning, that is, during the evaluation of policy requests generated for partial policy vectors. As the probability to derive an exact policy decision is higher for a partial policy request that contains more information, it is beneficial to know which updates will be done in future planner's world states in order to represent this information in the partial policy requests. A high-level effect construct will be introduced in this section in order to represent additions for the planner's world state that will be carried out during the execution of compound tasks and actions.

Definition 6.6. High-level effect for task TA is a positive literal $\tilde{L}$ such that during every possible execution of TA at least one literal L that refines $\tilde{L}$ or equal to it ($RefEql^l(L, \tilde{L})$) must be added into the planner's world state□

A set of high-level effects assigned to task TA in a current task network is returned by function $effects(TA)$. For each type of the planning tasks, the definition of high-level effects can be specified as follows:

- If a high-level effect $\tilde{L}_i$ is assigned to a primitive action TA_j, each operator that can be applied to TA_j should add to the planner's world state literal L_i such that $RefEql^l(L_i, \tilde{L}_i)$.
- If a high-level effect $\tilde{L}_i$ is assigned to a compound task or action TA_k, during each possible decomposition of TA_k an operator or compound action decomposition method should be executed that adds to the planner's world state literal L_i such that $RefEql^l(L_i, \tilde{L}_i)$.

When a task is decomposed during the planning, new tasks produced by the decomposition method are assigned with high-level effects that were specified for them by the method's author. In order to guarantee that for all literals assigned as high-level effects the corresponding properties are fulfilled, required correctness checks will be introduced into operators and methods execution routines (see Section 6.2.3). It should be noted that as high-level effects are preliminary effects specifications, they are introduced by the methods' authors optionally[8] to provide more information about future planner's world states for the policy evaluation.

Increasing and Decreasing effects sets

In order to take into account other possible modifications of the planner's world state, introduced by tasks and actions, Increasing and Decreasing effects sets will be introduced. For each task in a current task network, two sets of tasks are defined:

Definition 6.7. Necessary Precedents set $P_N(TA, TN)$ is a set of tasks in task network TN that must be executed before the task TA□

[8] Provided that checks that guarantee their correctness are satisfied.

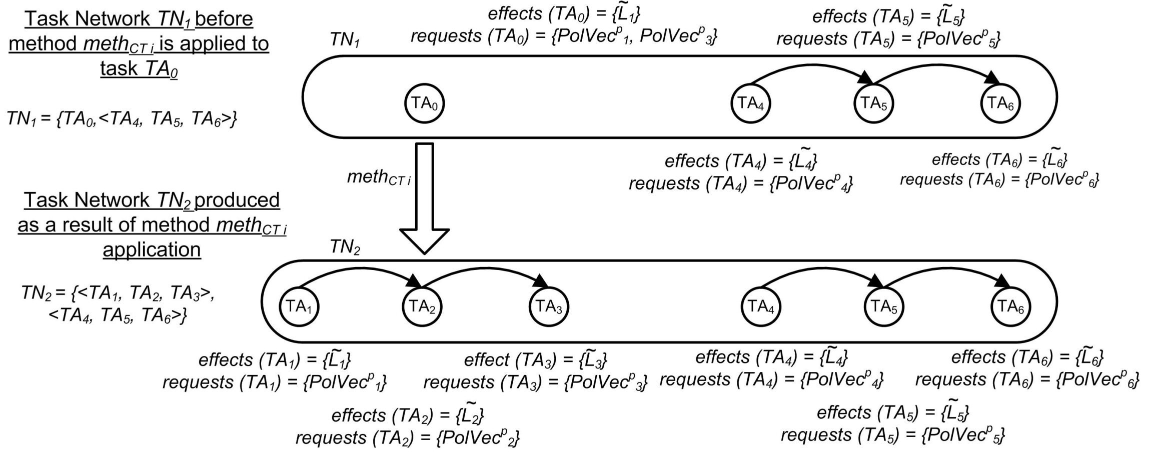

Figure 6.1: Example of compound task decomposition method $meth_{CT_i}$ execution

Definition 6.8. Possible Precedents set $P_P(TA, TN)$ is a set of tasks in task network TN that can be executed before the task TA (that are not planned after the task TA)□

As can be seen, for each task TA in a task network TN: $P_N(TA, TN) \subseteq P_P(TA, TN)$. For example, in Figure 6.1 in the lower task network TN_2 for task TA_3 necessary and obligatory precedents sets are: $P_N(TA_3, TN_2) = \{TA_1, TA_2\}$, $P_P(TA_3, TN_2) = \{TA_1, TA_2, TA_4, TA_5, TA_6\}$ (precedence relations are denoted using arrows).

In order to operate with possible effects that can be introduced into the planner's world state during the execution of tasks in a current task network, each task symbol TA_i^S within the planning domain is associated with the following sets.

Definition 6.9. Possible Negative Effects set $PosNegEff(TA^S)$ is a set of predicate symbols used in literals that can be removed from the planner's world state during the execution of a task (which can be a primitive/compound action or a compound task) with the task symbol TA^S□

Definition 6.10. Possible Positive Effects set $PosPosEff(TA^S)$ is a set of predicate symbols used in literals that can be inserted in the planner's world state during the execution of a task (which can be a primitive/compound action or a compound task) with the task symbol TA^S□

Sets $PosNegEff(TA^S)$ and $PosPosEff(TA^S)$ can be determined automatically using a recursive procedure, deriving the possible effects set for a compound task as a union of possible effects sets for tasks that can be produced as a result its decomposition, using any method applicable to it. For actions, possible effects sets are determined as a union of predicate symbols used in effects of any applicable operator[9]. For compound actions, which have effects and can be decomposed using methods, these two unions are, in turn, merged in order to produce the possible effects set.

[9]Obligations that can be generated for actions during the policy evaluation are also taken into account. A list of possible obligations for an action is retrieved from the obligation validation rules specified.

Correspondingly, when a method is applied, the possible effects set for the decomposed task is a superset relatively to the union of possible effects sets of all produced tasks or is equal to it.

Using the defined sets of predicate symbols and tasks, it is possible to derive the set of effects that can change the planner's world state before the execution of a task in the current task network. The specified sets take into account all possible ways how tasks in the task network can be executed.

Definition 6.11. Decreasing Effects (DE) Set $DE(TA, TN)$ is a set of predicate symbols used in literals that can be removed from the planner's world state before or during the execution of the task TA in the task network TN (see Formula 6.3)□

Definition 6.12. Increasing Effects (IE) Set $IE(T, TN)$ is a set of predicate symbols used in literals that can be added to the planner's world state before or during the execution of the task TA in the task network TN (see Formula 6.4)□

$$DE(TA, TN) = \cup_{TA_i \in P_P(TA,TN)} \{PosNegEff(TSim(TA_i))\} \cup PosNegEff^*(TSim(TA)) \quad (6.3)$$

$$IE(TA, TN) = \cup_{TA_j \in P_P(TA,TN)} \{PosPosEff(TSim(TA_j))\} \cup PosPosEff^*(TSim(TA)) \quad (6.4)$$

Increasing (or Decreasing) effects set for a task TA is derived as a union of Possible Positive Effects sets (or Possible Negative Effects sets, respectively) for tasks that can be executed before TA. The function $TSim(TA)$ returns a task symbol TA^S for the task atom TA, which is used to determine the possible effect sets. The sign '*' for Possible Positive Effects and Possible Negative Effects sets of the task TA denotes that these sets are included into the IE and DE sets only when TA is a compound action or task. IE and DE sets for the task TA are utilised to determine possible changes of the planner's world state that can happen before the evaluation of a partial policy request represented using a partial policy vector attached to the task TA. So when the task TA is a compound task, it is required to add possible effects of this task into the IE and DE sets as this policy request can be evaluated during the execution of some action produced as a result of TA decomposition. So before the evaluation of this policy request, any effect from the possible effects set for TA can be introduced. When the task TA is a primitive action, it is known that effects of this action will be applied after the evaluation of the policy request for this task, so they should not be included into the IE and DE sets for this primitive action.

6.2.2.2 Partial policy requests generation

Partial policy vectors, assigned to tasks in a current task network, are used for the construction of partial policy requests referring to future planner's world states. These policy requests are generated based on a planner's world state using the same procedure that was described for ordinary policy requests in Chapter 5. But for the construction of a partial policy request based on a partial policy vector attached to task TA_0 in a task network TN, instead of the

current planner's world state $CurState$ the planner's world state $State'$ is used. In this state, known updates that will be done before the execution of the task TA_0 are reflected: $State' = \{\cup_{TA_i \in P_N(TA_0,TN)}\{effects(TA_i)\} \cup CurState\}/L_{DES}$, i.e., high-level effects for tasks which necessarily will be executed before the task T_0 should be added to the current planner's world state $CurState$ and all literals L_{DES} that can be removed from the planner's world state are removed from $State$ and excluded from the high-level effects being added. Set L_{DES} is a subset of the set $\cup_{TA_i \in P_N(TA_0,TN)}\{effects(TA_i)\} \cup CurState$ and contains all literals from this set with predicate symbols in $\cup_{TA_j \in P_P(TA_0,TN)} DE(TA_j, TN)$[10]. Additionally, when during the construction of the partial policy request it is required to get properties and relations for a dummy object Dum_{ID} (for examople, when it is used as a designated object), they are retrieved from the Dummy Objects Space: $DumDescr(Dum_{ID})$.

So the partial policy request Req^p is a policy request (see Section 5.4.1) where a dummy object, a null property, a non-leaf hierarchical property-term or a loose time interval is used. A partial policy request can be evaluated correctly only taking into account the IE and DE sets for the task that its partial policy vector is attached to. These sets represent modifications that can be introduced before the execution of the considered action during the planning. So in a postponed policy enforcement, the input information for a policy request evaluation is represented as a *partial policy request tuple*: $PRtuple = \langle Req^p, IE(TA, TN), DE(TA, TN)\rangle$. According to the previous definitions, a partial policy request tuple $PPRtuple_i = \langle Req^p_i, IE(TA_i, TN_i), DE(TA_i, TN_i)\rangle$ is a refinement of a partial policy request tuple $PRtuple_j = \langle Req^p_j, IE(TA_j, TN_j), DE(TA_j, TN_j)\rangle$ if the following modifications were introduced to the $PRtuple_i$ tuple in comparison with the $PRtuple_j$ tuple:

- The partial policy vector is refined.
- Attribute values for the designated objects are refined if they are dummy objects.
- New properties and relation literals are added for dummy objects within the designated object contexts or existing properties and relation literals are refined for them.
- Properties and relations are added into the designated object contexts, if their predicate symbols are contained in $IE(TA_j, TN_j)$.

It should be noted that if a Dummy object is refined, refined versions of all its properties and relations contained in Req^p_j should be contained in Req^p_i. A fully known policy request tuple is a tuple that contains an ordinary policy request and the IE and DE sets are empty.

6.2.3 Operators and methods execution routines

According to the previous sections, extra activities should be carried out when methods and operators are executed during the postponed policy enforcement: high-level effects and partial policy

[10]Predicates that can be added into the planner's world state are processed using another mechanism, described in Section 6.3.

vectors should be assigned to tasks produced as a result of the decomposition and properties that guarantee correctness of this assignment should be checked.

First of all, definitions of methods and operators described in Section 5.3.2 are updated. When methods are specified, tasks in their task networks are provided with unique numbers used for their identification. So all tasks are specified as tuples $\langle Task^{ID}, TA\rangle$, where $Task^{ID}$ is a task number, TA is a task atom. Additionally, the definitions of compound action and compound task decomposition methods (will be designated as $meth$) are augmented with the following components (see 5.14), in order to specify which high-level effects and partial policy vectors assignments should be done within the current task network and which updates to the Dummy Objects Space should be introduced:

- $HLeffectsSpec(meth) = \{\ldots \langle Task_i^{ID}, \tilde{L}_i\rangle \ldots\}$ is a set of tuples containing a task number and a high-level effect. The high-level effects $\tilde{L}_i$ should be assigned to the tasks with $Task_i^{ID}$ during the decomposition.
- $PRVecSpec(meth) = \{\ldots \langle Task_j^{ID}, PolVec_j^p\rangle \ldots\}$ is a set of tuples containing a task number and a partial policy vector. The partial policy vectors $PolVec_j^p$ should be assigned to the tasks with $Task_i^{ID}$ during the decomposition.
- $DumStruc^{-}(meth) = \{\ldots \tilde{L}_m \ldots\}$ is a set of literals that should be removed from the Dummy Objects Space during the decomposition.
- $DumStruc^{+}(meth) = \{\ldots \tilde{L}_o \ldots\}$ is a set of literals that should be added into the Dummy Objects Space during the decomposition. The literals $\tilde{L}_o = p_o(\tau_{1o}, \ldots, \tau_{eo})$ should contain at least one term representing a dummy object: $\exists h \in \{1, 2, \ldots, e\}$ ($\tau_{ho} \in Term^{DumObj}$).

During the execution of methods, the specified activities should carried out. In addition to this, correctness checks should be carried out during the execution of operators and methods in order to guarantee that the required properties for the high-level effects, partial policy vectors and other components, introduced in the previous sections, are satisfied. These correctness checks, carried out during the decomposition of a compound action TA_0 into task network TN using a method $meth_{CA}^i$, are presented in Figure 6.2. High-level effects and partial policy vectors attached to task TA_0 should be refined by some (high-level) effects or (partial) policy vectors generated during the decomposition. Finally, properties and relations of dummy objects that were removed by this method from the Dummy Objects Space should be asserted into it in a refined form. When an operator or a compound task decomposition method is carried out, only relevant part of these checks should be performed. During the planning, substitution θ is used to store substitutions of dummy objects that have been carried out. It is updated when new substitutions are introduced and is utilised to check that the same dummy object-terms are refined to the same ordinary object-terms. When the planner backtracks, substitutions carried out during the cancelled operations are removed from this substitution.

For all high-level effects $\tilde{L}_j$ $(\tilde{L}_j \in effects(TA_0))$ check that:

$$\exists \langle Task_k^{ID}, \tilde{L}_k \rangle \in HLeffectsSpec(meth_{CA}^i) \text{ (} RefEql^l(\tilde{L}_k, \tilde{L}_j, \theta) \text{) or } \exists L_m \in effect^+(meth_{CA}^i) \text{ (} RefEql^l(L_m, \tilde{L}_j, \theta) \text{).}$$

For all partial policy vectors $PolVec_h^p$ $(PolVec_h^p \in requests(TA_0))$ check that:

$$\exists \langle Task_n^{ID}, PolVec_n^p \rangle \in PRVecSpec(meth_{CA}^i) \text{ (} RefEql^r(PolVec_n^p, PolVec_h^p, \theta) \text{) or } RefEql^r(polVector(meth_{CA}^i), PolVec_h^p, \theta)$$

For all removed dummy objects properties and relations $\tilde{L}_z$ $(\tilde{L}_z \in DumStruct^-(meth_{CA}^i))$ check that:

$$\exists \tilde{L}_x \in DumStruct^+(meth_{CA}^i) \text{ (} RefEql^l(\tilde{L}_x, \tilde{L}_z, \theta) \text{).}$$

Figure 6.2: Correctness checks for compound action decomposition method execution

These checks guarantee that during the planning all high-level effects, partial policy vectors and other constructs used for the partial policy requests generation are refined or remain constant. Additionally, during the planning, IE and DE sets for the same tasks within the current task network can only be reduced or remain unchanged. When an operator is applied, possible negative and positive effects sets for the action executed are excluded from the IE and DE sets of all tasks where this action was included in the Possible Precedents sets. When a method is executed, as it was stated before (see Section 6.2.2.1), a possible effects set for the task decomposed is a superset for a union of possible effects sets for all tasks added into the task network during the decomposition. Since in IE and DE sets, where the decomposed task has contributed to, its possible effects sets should be substituted by the union of possible effect sets of the tasks produced during the decomposition, these IEs and DEs cannot contain more elements than before the decomposition. Moreover, when a partial policy vector attached to some new task during the decomposition is connected using refinement or equal relation with a partial policy vector attached to the task decomposed, IE and DE sets for the new task should be included into the IE and DE sets for the decomposed task or be equal to them. This is true as IE and DE sets for the new task are equal to the IE and DE sets for the decomposed task, with an exception that in the expression for their calculation possible effects sets for the task decomposed are substituted by the union of possible effects sets for the new task and tasks from the generated task network that can be executed before it. So during an operator or method execution, partial policy request tuples created for partial policy vectors that remain the same during this planning step are refined or remain constant. Moreover, partial policy request tuples are refined or remain constant if they are created for partial policy vectors connected with the refinement or equal relation: one partial policy vector is attached to the task processed by the method (or operator) and another is attached to a task generated during the decomposition (or it corresponds to an action executed).

The method's constructs introduced in this section should be specified by the author of planning

domain based on his (or her) knowledge about possible execution traces for the tasks generated during the decomposition. The execution traces are determined by other methods and operators specified in the planning domain. A planning domain static analysis technique can be introduced in order to generate some high-level effects and partial policy vectors automatically, which can simply this specification task for the domain author. Based on the equality of task atoms' signatures, it is possible to predict which method or operator can be applied to which tasks. Variables used at the same positions can be propagated to lower level task atoms (and correspondingly to effects and policy vectors). Using this mechanism, it is possible to build possible decomposition traces for all tasks within the domain specification, based on which some signatures for high-level effects and partial policy vectors can be inferred. But this simple analysis would not give comprehensive results, since it does not include analysis of operator and method preconditions, which determine their applicability during the planning based on current variables values and perform variables instantiation. So, in any case, the results of this technique should be corrected by the domain author manually: at least he (or she) should put dummy objects, null properties and hierarchical properties instead of the variables in the inferred signatures, which were not mapped to task atom's variables. Additionally, he (or she) can significantly extend them, manually analysing the methods and operators: some variables can be instantiated, dummy objects description can be added. Also as a set of operators and methods that can be utilised during an execution of a task can be limited based on the analysis of its preconditions and preconditions of higher-level tasks, more high-level effects and partial policy vectors can be specified.

6.2.4 Postponed policy enforcement

A postponed policy enforcement is based on the evaluation of partial policy request tuples created for policy vectors attached to tasks in a current task network during the planning. A partial policy evaluation algorithm was designed (see Section 6.3), which can correctly process partial information in these tuples. As a partial policy requests tuple contains only known information and can be modified later on during the planning, an exact policy decision for some partial policy requests cannot be determined. In order to represent such situation, an additional policy decision for the XACML policy language was introduced.

Definition 6.13. Indeterminate Temporal decision is an additional decision introduced for the XACML policy language. It indicates that none of decisions in set M_1, containing the standard XACML policy decisions $\mathbf{M}_1 = \{P, D, Ind, N/A\}$, can be produced during the evaluation of a partial policy request tuple because not all required information is available□

The set of standard XACML policy decisions M_1 will be denoted as Permanent policy decisions set. Obviously, during the evaluation of partial policy requests, permanent decisions can also be produced, even if the request is specified partially. For example, available information can be

sufficient to infer exact decisions for all applicable policies, or Indeterminate Temporal decisions, produced during the evaluation, can be overpowered by permanent decisions during the policies or rules combining. Obviously, when a policy request tuple is fully known, a permanent decision should be produced for it.

The Indeterminate Temporal decision is distinct from the Indeterminate decision within the Permanent policy decisions set. The latter decision indicates that there was an error during the evaluation that prevents a policy engine to produce a Permit or Deny decision. This decision is generated when policies are specified incorrectly or when data with wrong data type are retrieved from the policy request. The difference with Indeterminate Temporal can be illustrated with the following example. Assume that during the policy evaluation some obligatory attribute should be retrieved from the policy request, but this attribute is missing there. The standard Indeterminate decision should be produced in this situation if it is known that more values cannot be added for this attribute as a result of the partial policy request refinement. Indeterminate Temporal should be produced when new values can be added. So the standard Indeterminate decision should be returned when it is known that further refinement of the partial policy request tuple will not lead to a substitution of this decision by a permanent policy decision. In order to distinguish two Indeterminate decisions, a new set of policy decisions is introduced: M_1^p (see Formula 6.5). The standard Indeterminate decision from the set M_1 will be referred as '*Indeterminate Permanent*' decision.

$$\mathbf{M_1^p} = \{P, D, IndPerm, IndTemp, N/A\} \tag{6.5}$$

When the postponed policy enforcement is utilised during the planning, after the execution of each method and operator a special routine is carried out that evaluates partial policy requests for vectors attached to tasks within the current task network. The underlying principle of the postponed policy enforcement, which makes possible enforcement of decisions produced as a result of the partial policy requests evaluation, is now briefly explained. Any permanent decision returned during the evaluation of a partial policy request tuple should persist when this tuple is refined. That is, for any partial policy request tuple refining it, the same permanent decision should be returned. Based on this fact, at each planning step decisions produced as a result of partial policy requests evaluation can be enforced by the planner: when a request is permitted the planning can go forward; when it is denied a backtrack should be done; when an Indeterminate Temporal is received the request should be postponed and re-evaluated later, when more information is available. Partial policy request tuples evaluated at some planning step are refined at the subsequent steps of the planning. Hence, if Permit is received as a result of the partial policy vector evaluation, when at the next planning step the same vector or a vector refining it[11] is evaluated Permit decision

[11] It is a vector assigned to a task produced during the decomposition or evaluated during the operator execution that is connected with the refinement or equal relation with the permitted vector and .

Evaluate (old task network TN', old state $State'$, new task network TN'', new state $State''$)

1. Determine set of partial policy vectors S_{PolVec^p} that are attached to tasks in TN'' and do not have '*Permit*' labels
2. **Loop** for every $PolVec_i^p \in S_{PolVec^p}$:

   2.1. Construct sets $IE(T'',TN'')$ and $DE(T'',TN'')$ for task T'' (a task that $PolVec_i^p$ is attached to) using formulas 6.3, 6.4

   2.2. **If** partial policy request tuple for $PolVec_i^p$ can produce new results, considering TN' and $State'$ updates **then** evaluate it using the partial policy evaluation algorithm (see Section 6.3). Process the decision returned:

   •*Permit or Not Applicable.* Add '*Permit*' label to partial policy vector $PolVec_i^p$ in task network TN''

   •*Deny or Indeterminate Permanent.* **Return** Failure

   •*Indeterminate Temporal.* Carry on evaluation

   endif

   End loop
3. **Return** Success

Figure 6.3: Partial policy requests evaluation algorithm

will also be returned. In order to avoid superfluous policy evaluations, partial policy vectors for which Permit decisions were produced are marked with '*Permit*' label and are not evaluated at the subsequent planning steps. Additionally, these '*Permit*' labels are propagated to partial policy vectors that refine the permitted vectors.

An algorithm carried out after each planning step, when the postponed policy enforcement is used, is presented in Figure 6.3. During the execution of this algorithm, first of all, partial policy vectors attached to tasks in a current task network are selected for the execution. Then, partial policy request tuples are constructed for them and evaluated. Results of their evaluation are analysed and processed as this was described earlier. When a failure is produced by this algorithm, backtracking is required. Using this algorithm, during the postponed policy enforcement it is guaranteed that all policy request vectors are evaluated into permanent policy decisions eventually. Moreover, during the postponed policy enforcement, fully specified policy vectors, that should be evaluated when an operator or a compound action decomposition method is carried out, are evaluated only if they do not have '*Permit*' labels assigned to them. As can be noted, this algorithm has a restricted functionality as it does not support all policy evaluation outcomes utilised in policy-based planning (see Chapter 5). In concrete, it does not support policy obligations and conditions. Correspondingly, in order to resolve this restriction, this routine and the introduced partial policy evaluation algorithm should be updated for their support[12].

[12] As will be shown in Chapter 7, in the CEP generation problem, the postponed policy enforcement is utilised at the planning stage when obligations and conditions are not used. So this restriction does not limit the usage of the

During the execution of this routine, it is possible to evaluate all partial policy vectors without "*Permit*" labels. But in order to reduce the number of evaluations, only vectors for which partial policy tuples were updated during the last planning step are evaluated. A partial policy vector $PolVec_i^p$, attached to a task TA'' in task network TN'' is evaluated if the planner's world state is updated or if this is a new partial policy vector, added during the last planning step, or if the vector $PolVec_i^p$ refines a partial policy vector in TN'[13] that was attached to the task processed during the last planning step, or if one of the following conditions is satisfied. These conditions should be satisfied for the task TA'' in TN'' and a task TA' in TN' that has a policy vector corresponding to $PolVec_i^p$: this vector should be equal to $PolVec_i^p$ and be attached to the same task ($TA' = TA''$) or the task processed during the last planning step:

1. A set of high-level effects for tasks in $P_N(TA'', TN'')$, in comparison with the corresponding set for tasks in $P_N(TA', TN'')$, contains new high-level effects or some high-level effects are refined.
2. For IE and DE sets: $IE(TA'', TN'') \subset IE(TA', TN')$ or $DE(TA'', TN'') \subset DE(TA', TN')$.

If the planner's world state was not changed, a partial policy vector copied to a new task produced during the decomposition should not be re-evaluated, if these conditions are not satisfied for it. For example, condition two is satisfied if a partial policy vector is copied to a task TA within the task network $network(meth)$, generated by during the decomposition of method $meth$, and there is some task in $network(meth)$ that should be executed before TA and that has some high-level effects assigned. In Figure 6.1, such vector is $PolVec_3^p$ as the high-level effects $\tilde{L}_1$ and $\tilde{L}_2$ are introduced for the tasks TA_1 and TA_2 that should be executed before the task TA_3, which has $PolVec_3^p$ attached. Also these conditions are taken into account for partial policy vectors that are not modified during some planning step. For example, in Figure 6.1, when task TA_0 is decomposed using method $meth_{CTi}$, partial policy requests for tasks TA_4, TA_5 and TA_6 should be evaluated only if set IE or DE is reduced for them.

6.3 Partial policy evaluation

6.3.1 Requirements to partial policy evaluation

A partial policy request tuple introduced at some planning step is refined at further planning steps, until at some planning step a fully known policy request tuple is generated. After each refinement, the partial policy request tuple is evaluated (if this is required). A schema in Figure 6.4 shows how partial policy evaluations are carried out during the planning. A partial policy evaluation should satisfy the following property that makes it possible to enforce decisions produced as a result of partial policy evaluation during the planning. This is the *permanent decisions preservation*

postponed policy enforcement for the CEP construction.

[13] It is the task network before the last operator or method execution.

property: if a permanent decision from the set $\mathbf{M_1}$ is produced for a partial policy request tuple, it cannot be changed to any other decision when the partial policy request tuple is refined. The refinement or equal relation for partial policy request tuples, introduced in the previous section, is a partial order relation on the set of all possible tuples $PRtuple$, which will be denoted as $\sqsubseteq$. On the set of policy decisions M_1^p, a partial order representing possible changes of policy decision as a result of the policy tuple refinements can also be introduced as represented in Figure 6.5. This order is based on the fact that an $IndTemp$ decision can be substituted by any permanent decision when the partial policy request tuple is refined. This partial order will be referred as the approximation order: it represents the 'less defined than or equal' relation for the amount of information carried by different policy decisions ($IndTemp$ carries less information than any other decision). If we represent a partial policy evaluation of a policy set as a function $ParEvaluate^{PS}$: $PolicySet \times PRtuple \to M_1^p$, then the permanent decisions preservation property is formalised as a monotonicity of this function defined in terms of the partial orders on sets $PRtuple$ and M_1^p (provided that policy set $[\![$ `PolicySet`$]\!]$ is constant):

$$PRtuple_i \sqsubseteq PRtuple_j \to \tag{6.6}$$
$$ParEvaluate^{PS}([\![\texttt{PolicySet}]\!], PRtuple_i) \sqsubseteq ParEvaluate^{PS}([\![\texttt{PolicySet}]\!], PRtuple_j)$$

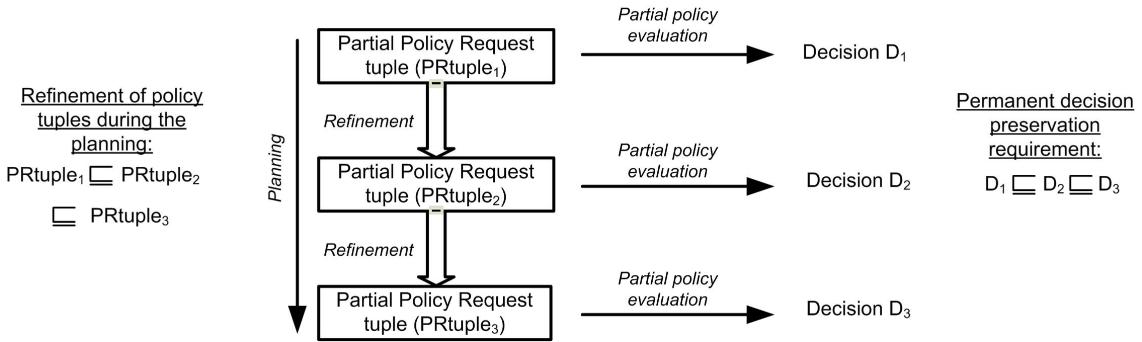

Figure 6.4: Partial policy evaluation process

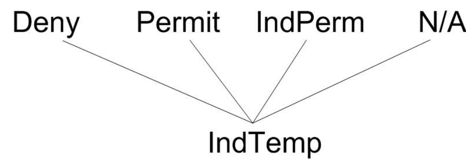

Figure 6.5: Approximation order on set M_1^p

When the postponed policy enforcement is used, the partial policy evaluation mechanism is used instead of the standard XACML policy evaluation, even if a fully known policy request tuple is evaluated. So the partial policy evaluation should be an extension of the standard XACML policy

evaluation. If all required information is available during the evaluation, the outcomes of the partial policy evaluation should be equal to the outcomes of the standard policy evaluation mechanism. Suppose, we introduce a function $evaluate^{PS}_{ext}$ that extends the ordinary policy evaluation function $evaluate^{PS}$ (see Section 4.5.2) and returns decision $IndTemp$ if input policy request tuple is not fully known. Then, this requirement can be formalised as follows: function $ParEvaluate^{PS}$ should refine function $evaluate^{PS}_{ext}$ (provided that policy set $[\![\texttt{PolicySet}]\!]$ is constant): $evaluate^{PS}_{ext} \sqsubseteq ParEvaluate^{PS}$. That is, for all $PRtuple_i$:

$$evaluate^{PS}_{ext}([\![\texttt{PolicySet}]\!], PRtuple_i) \sqsubseteq ParEvaluate^{PS}([\![\texttt{PolicySet}]\!], PRtuple_i) \tag{6.7}$$

So the partial policy evaluation must have the following properties:

- The permanent decision preservation (i.e., the function $ParEvaluate^{PS}$ should be monotonic).
- It should be an extension of the standard XACML policy evaluation (i.e., the function $evaluate^{PS}_{ext}$ should be approximation of $ParEvaluate^{PS}$).

Another desired property of the partial policy evaluation, which improves the planning performance and reduces the planning time, is the following. Based on a partial policy request tuple a partial policy evaluation mechanism should tend to return a permanent decision when it is possible (i.e., when it does not contradict to the requirements described above). The motivation of this requirement is that only permanent decisions lead to performance gains during the postponed policy enforcement.

In Chapter 4 it was shown how a policy set evaluation function $evaluate^{PS}$ can be specified as a composition of functions (a structure for this composition is determined by a policy set being evaluated). In order to define the partial policy evaluation function $ParEvaluate^{PS}$, it is assumed that the XACML policy specification mechanism remains the same. So policies in the partial policy evaluation can also be formally represented as a composition of functions (the same principles of how to infer a composition of functions based on a policy set specified are used). But functions used in these compositions in order to represent the policy constructs should be updated. These functions should be defined on extended domains, which can additionally represent incomplete information, and define how values added to these domains should be evaluated. In the following sections these functions are described. In order to guarantee that the monotonicity and extension requirements are satisfied for the function $ParEvaluate^{PS}$, it is can be shown that they are satisfied for all its constituent functions, as a composition of monotonic functions is monotonic and two compositions of monotonic functions are related with approximation relation if they have the same structure and all functions in one composition are approximations of the corresponding functions in another composition. The former statement is a known theorem in the order theory

Table 6.1: Table of values for Policy evaluation function P^{ep}

Target \ Combining decision	P	IndPerm	IndTemp	D	N/A
False	N/A	N/A	N/A	N/A	N/A
IndPerm	IndPerm	IndPerm	IndPerm	IndPerm	IndPerm
IndTemp	IndTemp	IndTemp	IndTemp	IndTemp	IndTemp
True	P	IndPerm	IndTemp	D	N/A

(e.g., in [22]), whereas the later statement can be easily proven using transitive property of orders. The described updates of the functions do not specify how loose time intervals are processed, as their processing will be described separately in Section 6.3.7.

6.3.2 Policy set evaluation function

The auxiliary policy set evaluation function $P^e : TRVal \times M_1 \to M_1$, which was introduced to define the policy set evaluation routine (see Chapter 4), processes the truth value returned from the policy set target and a decision in the set M_1 returned from the policy combining (see Table 4.1). For a partial policy evaluation, a new function $P^{ep} : TRVal^p \times M_1^p \to M_1^p$ that defines the policy set evaluation routine was introduced. Its input policy combining decision, as well as, its output decision are members of the set M_1^p, the set of decisions for the partial policy evaluation. The truth values set $TRVal$, which is a co-domain of the target evaluation function, was extended by value $IndTemp$, referring to a situation when not all required information is available in the partial policy request tuple in order to produce a permanent truth value. But, in contrast to the Indeterminate Permanent truth value, which is represented as $IndPerm$ in this set, for $IndTemp$ it is known that it should be refined into a permanent truth value ($True$, $False$ or $IndPerm$) when the policy request tuple is refined into a fully known policy request tuple. So $TRVal^p = \{True, False, IndTemp, IndPerm\}$. Only one 'less defined' value $IndTemp$ is contained in this set, so its partial order of approximation is flat. It can be defined using the following orderings: $IndTemp \sqsubseteq True$, $IndTemp \sqsubseteq False$, $IndTemp \sqsubseteq IndPerm$. Similarly to the order defined for the set M_1^p, this partial order shows how a truth value can be changed when the partial policy request tuple is refined.

The table of values for the function P^{ep} is presented in Table 6.1. According to the XACML specification, a policy set should be evaluated only if its target is true. Hence, in rows where the target is equal to $False$, $IndPerm$ or $IndTemp$, the policy combining value has no influence and even is not evaluated. When the target is $IndTemp$, this means that a permanent decision cannot be produced for this partial policy request tuple. In this case, $IndTemp$ is returned by P^{ep}, because the resultant permanent policy decision will depend entirely on a future target evaluation result. The situation when a target is $True$ but policy combining decision is $IndTemp$ is equivalent and $IndTemp$ should also be returned.

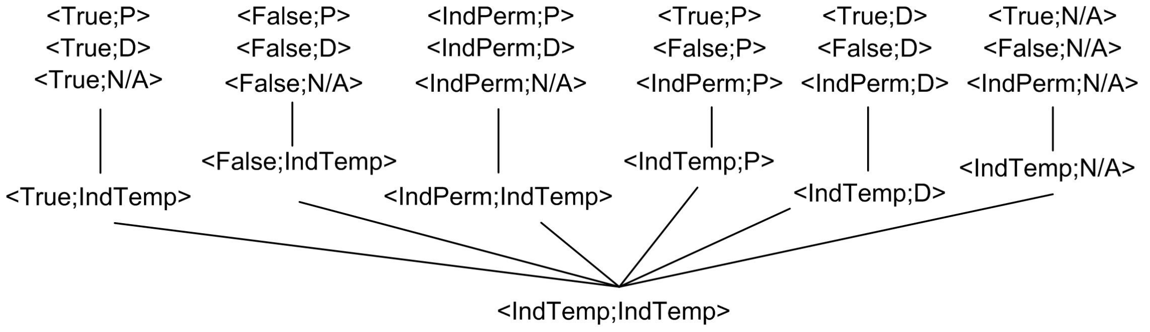

Figure 6.6: Approximation order on $TRVal^p \times M_1^p$ set

In order to check the monotonicity of the P^{ep} function, it is required to check that during each possible modification of its argument values, such that new argument values refine old values, its resultant value is also changed from a less defined value to a more defined one (or it is constant). As the function P^{ep} has two arguments, an approximation order that can be used to represent refinement relations on its arguments is an approximation order on the Cartesian product of their domains. Such order for product $TRVal^p \times M_1^p$ is shown in Figure 6.6. In order to check the monotonicity using the table of values, it is required to analyse each value produced by the function when it has $IndTemp$ in (at least) one of its arguments. If $IndTemp$ is a value for the left argument, the output value should be less defined than every other value in the same column of the table or it can be equal to it (if $IndTemp$ is in the right argument, the row values are analysed). As can be seen, this requirement is satisfied. The second requirement that P^{ep} should be an extension of P^e is easily checked. If the column and the row representing input $IndTemp$ arguments are removed, the table of values for P^{ep} is transformed into one for P^e (with the introduced re-naming of Ind decisions and truth values).

6.3.3 Policy and policy set combining algebras

Policy combining operations that were introduced to formalise the permit- and deny-overrides policy combining algorithms are operations $\bullet_p^{PO}$ and $\bullet_p^{DO}$. Properties of algebras, which they form on the set of policy decisions M_1, were analysed in Section 4.5.2. In order to use these operations in the partial policy evaluation, they were extended to the set M_1^p. These new operations are designated as $\bullet_{pp}^{PO}$ and $\bullet_{pp}^{PO}$. Tables of values for them are presented in Table 6.2. Properties of algebraic structures with the $\bullet_p^{PO}$ and $\bullet_p^{DO}$ operations (see Section 4.5.2) should be preserved for new operations in order to keep the essence of these combining algorithms. Hence, elements Permit P and Deny D should be adsorbing elements for operations $\bullet_{pp}^{PO}$ and $\bullet_{pp}^{DO}$ correspondingly. In natural orders defined by the permit- and deny-overrides, the new $IndTemp$ decision should be situated immediately after the adsorbing decision. In this case, the highest priority for the adsorbing decision is preserved and the new operations are monotonic. Assume that we place

$IndTemp$ to another position in the natural order for the permit-overrides and when $IndTemp$ is combined with some other decision X (excepting P) decision X is returned as a result. If in future input $IndTemp$ is refined into P, decision X must be changed into P, as it is the absorbing element, what contradicts with the monotonicity property.

Table 6.2: Tables of values for permit- and deny-overrides policy combining operations $\bullet_{pp}^{PO}$ and $\bullet_{pp}^{DO}$

$\bullet_{pp}^{PO}$	P	IndTemp	IndPerm	D	N/A
P	P	P	P	P	P
IndTemp	(1) P	(2) IndTemp	IndTemp	IndTemp	IndTemp
IndPerm	P	(3) IndTemp	IndPerm	IndPerm	IndPerm
D	P	(4) IndTemp	IndPerm	D	D
N/A	P	(5) IndTemp	IndPerm	D	N/A

$\bullet_{pp}^{DO}$	D	IndTemp	IndPerm	P	N/A
D	D	D	D	D	D
IndTemp	D	IndTemp	IndTemp	IndTemp	IndTemp
IndPerm	D	IndTemp	IndPerm	IndPerm	IndPerm
P	D	IndTemp	IndPerm	P	P
N/A	D	IndTemp	IndPerm	P	N/A

Permit- and deny-overrides policy combining operations for the partial policy evaluation $\bullet_{pp}^{PO}$ and $\bullet_{pp}^{DO}$ have the same properties as the ordinary permit- and deny-overrides policy combining operations (see Section 4.5.2): they are commutative, idempotent and associative. So the corresponding magmas $\mathcal{L}_{pp}^{PO} = \langle M_1^P, \bullet_{pp}^{PO} \rangle$ and $\mathcal{L}_{pp}^{DO} = \langle M_1^P, \bullet_{pp}^{DO} \rangle$ are semigroups and semilattices. The natural orders for them are extensions of natural orders for semilattices of the ordinary policy combining operations. Therefore they are not duals, so the algebra that contains these two semilattices is not a lattice. As extra decisions are inserted to the central parts of the natural orders, the designated elements of the algebra $\mathcal{A}_p$ are preserved in the corresponding algebra $\mathcal{A}_{pp} = \langle \mathbf{M_1^P}, \{\bullet_{pp}^{PO}, \bullet_{pp}^{DO}, N/A, P, D\} \rangle$.

In order to show that operations $\bullet_{pp}^{PO}$ and $\bullet_{pp}^{DO}$ are monotonic, it is required to trace updates of operation outcomes when their operands are refined according to the approximation order for the Cartesian product $M_1^p \times M_1^p$ (it can be constructed similarly to the order for $TRVal^p \times M_1^p$ product in Figure 6.6). Outcomes of the operations should be updated according to the approximation order on the M_1^p set. As these operations are symmetric, it is sufficient to check these properties only for values below or above the diagonal of their tables of values. We consider values below or at the diagonal of the table for the operation $\bullet_{pp}^{PO}$ (see Table 6.2). Cases where operands of this operation can be refined are denoted using numbers in brackets. As can be seen, when $IndTemp$ is changed to any other decision for these cases, the resulting decision is either constant or is updated from $IndTemp$ to a permanent decision. Hence, the monotonicity property is satisfied (analogously, for the deny-overrides operation it is also satisfied). The requirement concerning the extension of

operations $\bullet_p^{PO}$ and $\bullet_p^{DO}$ is also satisfied.

6.3.4 Policy and rule evaluation functions

During the policy evaluation, the truth value returned when a target is evaluated and the decision, produced during the rule combining, are processed and a decision in the set M_1^p is produced. As was specified in Section 4.5.3, rule combining results, as well as the rule evaluation results, are specified using decisions in the set M_2 (different from the policy evaluation decision set M_1), because it is required to distinguish which effect a rule could return if Indeterminate did not occur. For the partial policy evaluation, set M_2^p is defined based on the set M_2, similarly as the set M_1^p was defined. New decisions, corresponding to the Indeterminate Temporal decision, are introduced. As it is required to distinguish possible rule effects, two such decisions were added (see Formula 6.8). The approximation order on the set M_2^p is presented in Figure 6.7(A.) and represents possible updates of rule decisions during the policy request tuple refinement.

$$M_2^p = \{P, D, P(IndPerm), D(IndPerm), P(IndTemp), D(IndTemp), N/A\} \tag{6.8}$$

Similarly with the ordinary policy evaluation (see Section 4.5.3), in the partial policy evaluation the function P^{ep}, which was utilised for policy sets, is used to formalise the policy evaluation also. Additionally, function f^p is defined to transform a decision in the set M_2^p, produced during the rules combining, into the set M_1^p (see Figure 6.7 (B.)). Considering the defined approximation orders on the sets M_2^p and M_1^p, this function is obviously monotonic, because it maps both $P(IndTemp)$ and $D(IndTemp)$ decisions into the Indeterminate Temporal decision in M_1^p.

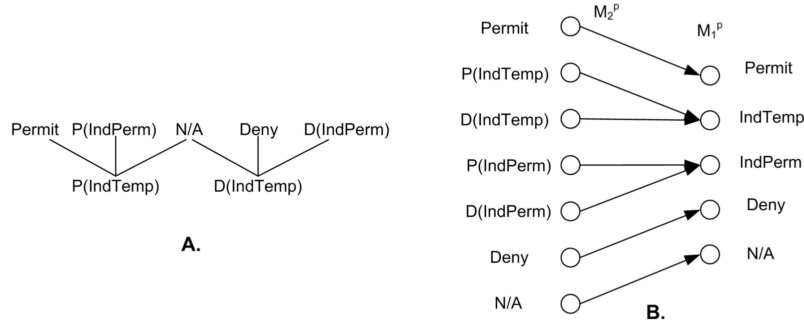

Figure 6.7: Approximation order on set M_2^p (A.) Graph for decisions mapping function $f_p : M_2^p \to M_1^p$ (B.)

The rule evaluation function for the partial policy evaluation $R^{ep} : EffectSpec \times TRVal^p \times TRVal^p \to M_2^p$ processes truth values generated as a result of the target and condition evaluation and considers the effect (Deny or Permit) in the $Effect$ part of the rule. The table of values for function R^{ep} (see Table 6.3) indicates that it is an extension of the rule evaluation function R^e (see Section 4.5.3). If the Indeterminate Temporal decisions are removed, the table of values for

Table 6.3: Table of values for rule evaluation function R^{ep}

Target \ Condition	False	IndPerm	IndTemp	True
False	N/A	N/A	(5) N/A	N/A
IndPerm	D(IndPerm) or P(IndPerm)	D(IndPerm) or P(IndPerm)	D(IndPerm) or (6) P(IndPerm)	D(IndPerm) or P(IndPerm)
IndTemp	D(IndTemp) or (1) P(IndTemp)	D(IndTemp) or (2) P(IndTemp)	D(IndTemp) or (3) P(IndTemp)	D(IndTemp) or (4) P(IndTemp)
True	N/A	D(IndPerm) or P(IndPerm)	D(IndTemp) or (7) P(IndTemp)	D or P

the ordinary rule evaluation function R^e will be obtained (considering the introduced re-namings). As the evaluation of a rule is similar with the policy evaluation, this function was defined using the same principles: condition should be evaluated and contribute to the result decision only when the target is evaluated to True (see Section 4.5.3).

The monotonicity of the function R^{ep} can be evaluated if an order defined on the Cartesian product of sets $TRVal^p \times TRVal^p$ is taken as domain order (it is presented in Appendix A in Figure A.1) and the order in Figure 6.7(A.) is taken as the order for function results. A specific decision used in the rule effect part is not considered, as Indeterminate decisions with different effects (e.g., $P(IndTemp)$ and $D(IntTemp)$) are related with the same ordering structure in M_2^p. All values in the table of values for R^{ep} should not be checked, as permanent decisions cannot change their values. Only values of the function produced when at least one of its arguments is $IndTemp$ should be considered (they are denoted with numbers in brackets in Table 6.3)[14]. All the possible changes of function values based on possible modifications of input values that conform with the order on $TRVal^p \times TRVal^p$ set are analysed separately in Appendix A as Formulae A.1 - A.7. Based on the results of this analysis it can be concluded that function R^{ep} is monotonic. For example, if a value of condition is $IndTemp$ and the value of target is not True, any permanent Truth value that could substitute $IndTemp$ value in the condition will not change the resultant rule decision (see cases 5 and 6 and Formulae A.5, A.6). Hence, the condition value should be ignored and a permanent decision, determined based on the target value, should be returned. For cases 1 - 4, when the target of a rule is evaluated as $IndTemp$, $IndTemp$ is also returned by the rule evaluation function. This is due to the fact that the value of target always determines the overall result of the rule evaluation. Even if the value of a condition is a permanent decision, this does not give the possibility to infer a permanent decision for the rule. The monotonicity in these cases is assured, as $IndTemp$ decision returned can be refined into any permanent decision (see Formulae A.1 - A.3).

[14]As was stated earlier, we consider only values situated below or at the diagonal of the table.

Table 6.4: Table of values for permit-overrides and deny-overrides rule combining operations $\bullet^{PO}_{rp}$ and $\bullet^{DO}_{rp}$

$\bullet^{PO}_{rp}$	P	P(IndT)	P(IndP)	D	D(IndT)	D(IndP)	N/A
P	P	P	P	P	P	P	P
P(IndT)	(1) P	(2) P(IndT)	P(IndT)	P(IndT)	P(IndT)	P(IndT)	P(IndT)
P(IndP)	P	(3) P(IndT)	P(IndP)	P(IndP)	P(IndP)	P(IndP)	P(IndP)
D	P	(4) × P(IndT)	P(IndP)	D	D	D	D
D(IndT)	(8) P	(5) × P(IndT)	(9) P(IndP)	(10) D	(11) D(IndT)	D(IndT)	D(IndT)
D(IndP)	P	(6) × P(IndT)	P(IndP)	D	(12) D(IndT)	D(IndP)	D(IndP)
N/A	P	(7) P(IndT)	P(IndP)	D	(13) D(IndT)	D(IndP)	N/A

$\bullet^{DO}_{rp}$	P	P(IndT)	P(IndP)	D	D(IndT)	D(IndP)	N/A
P	P	P	P	P	P	P	P
P(IndT)	P	P(IndT)	P(IndT)	P(IndT)	P(IndT)	P(IndT)	P(IndT)
P(IndP)	P	P(IndT)	P(IndP)	P(IndP)	P(IndP)	P(IndP)	P(IndP)
D	P	P(IndT)	P(IndP)	D	D	D	D
D(IndT)	P	P(IndT)	P(IndP)	D	D(IndT)	D(IndT)	D(IndT)
D(IndP)	P	P(IndT)	P(IndP)	D	D(IndT)	D(IndP)	D(IndP)
N/A	P	P(IndT)	P(IndP)	D	D(IndT)	D(IndP)	N/A

6.3.5 Rule combining algebras

Rule combining algorithms were formalised as operations $\bullet^{PO}_{r}$ and $\bullet^{DO}_{r}$ and properties of corresponding algebras were analysed in Section 4.5.4. In the partial policy evaluation, permit-overrides and deny-overrides rule combining operations $\bullet^{PO}_{rp}$ and $\bullet^{DO}_{rp}$, defined on set M^p_2, should extend these operations and preserve their properties. Tables of values for operations $\bullet^{PO}_{rp}$ and $\bullet^{DO}_{rp}$ are presented in Table 6.4. Absorbing elements in these operations should be preserved. Hence, when decision $P(IndTemp)$ or $D(IndTemp)$ is combined with an absorbing element, this adsorbing element can be returned as a result of the operation (e.g., in permit-overrides, if one operand is $P(IndTemp)$ or $D(IndTemp)$ and another is Permit, when the former operand is refined into another decision the result of the operation is guaranteed to be Permit). Additionally, when the permit-overrides is used, if $P(IndTemp)$ is combined with any other decision excepting *Permit* $P(IndTemp)$ should be returned. If we assume that $P(IndTemp)$ operand is refined into Permit, the result of the operation should also be refined into Permit, but the only decision that can be refined into Permit is $P(IndTemp)$. Analogously, since $D(IndTemp)$ can be refined to Deny, when it is combined with N/A or $D(IndPerm)$, $D(IndTemp)$ should be returned (since Deny is an absorbing element for these decisions and the only decision that can be refined into Deny is $D(IndTemp)$).

Permit-overrides and deny-overrides rule combining operations have the same properties, as the

corresponding operations in ordinary policy evaluation (see Section 4.5.4): they are commutative, idempotent and associative. Natural orders for these operations are specified in Formulae 6.9 and 6.10. These orders are not duals, so an algebra containing two these operations is not a lattice. As new decisions were added into the central parts of these orders, their identity and absorbing elements are the same: N/A is the identity element, P and D are absorbing elements respectively for permit-overrides and deny-overrides.

$$N/A \leqslant_{rp}^{PO} D(IndP) \leqslant_{rp}^{PO} D(IndT) \leqslant_{rp}^{PO} D \leqslant_{rp}^{PO} P(IndP) \leqslant_{rp}^{PO} P(IndT) \leqslant_{rp}^{PO} P \tag{6.9}$$

$$N/A \geqslant_{rp}^{DO} P(IndP) \geqslant_{rp}^{DO} P(IndT) \leqslant_{rp}^{DO} P \leqslant_{rp}^{DO} D(IndP) \geqslant_{rp}^{DO} D(IndT) \geqslant_{rp}^{DO} D \tag{6.10}$$

For the monotonicity analysis of operations $\bullet_{rp}^{PO}$ and $\bullet_{rp}^{DO}$, the approximation order on the set $M_2^p \times M_2^p$ should be used to define possible operands refinements. The approximation order on the set M_2^p represents possible result decision refinements. As these operations are symmetric, it is sufficient to analyse only one triangle in their tables of values. For operation $\bullet_{rp}^{PO}$, cases when operands of the operation can be refined are presented using numbers in brackets in its table of values. Proofs of the monotonicity property for these cases are presented in Appendix A. As operation $\bullet_{rp}^{DO}$ was defined by analogy, the same results are expected for it. The monotonicity property is not satisfied for some of the cases, which are designated using the '$\times$' symbol in Table 6.4 and frames in Appendix A. In these cases, a Deny-based decision (i.e., Deny, $D(IndTemp)$, $D(IndPerm)$) is combined with $P(IndTemp)$ and, as a result, $P(IndTemp)$ is returned. But the operand with $P(IndTemp)$ decision can be refined into a N/A decision, and N/A is the identity element for this operation. Hence, after such refinement, the operation will return the Deny-based decision that was used as an input and that the $P(IndTemp)$ decision cannot be refined to (because $P(IndTemp)$ can be refined only to a Permit-based decision and N/A). Hence, the operations $\bullet_{rp}^{PO}$ and $\bullet_{rp}^{DO}$ are not monotonic.

But it can be shown that the non-monotonicity of the permit-overrides and deny-overrides operations does not lead to the non-monotonicity of the overall policy set evaluation function. This is due to the fact that a composition of two non-monotonic functions can still be monotonic. As was shown in Section 6.3.4, the result of rule combining in the set M_2^p is mapped to the set M_1^p using the function f_p, before it can be processed by the policy evaluation function P^{ep}. Hence, it is possible to analyse the monotonicity of the function composition $f_p \circ \bullet_{rp}^{PO}$ that has domain $M_2^p \times M_2^p$ and co-domain M_1^p. If this composition is monotonic, then the overall monotonicity of the policy evaluation is not broken.

The proofs of the monotonicity for composition $f_p \circ \bullet_{rp}^{PO}$ are presented in Appendix A. Only the cases where the non-monotonic behaviour of the operation was found were considered (for other cases the monotonicity is assured at the rules combining level). It was shown that for all three

cases of $\bullet_{rp}^{PO}$ non-monotonicity when composition $f_p \circ \bullet_{rp}^{PO}$ is considered instead of $\bullet_{rp}^{PO}$, output values become monotonic. These facts are explained by the nature of the f_p function that eliminates bindings to Permit and Deny effects for Indeterminate decisions. Hence, in cases that were rejected because the $P(IndTemp)$ decision produced cannot be refined into a Deny-based decision in the set M_2^p, result decision $P(IndTemp)$ is mapped into $IndTemp$, which can be refined into any other decision in set M_1^p, according to its approximation order.

6.3.6 Target and condition evaluation functions

6.3.6.1 Information retrieval from partial policy request tuples

In the standard XACML policy evaluation, all information retrieved from a policy request is represented as Bags of values. When the partial policy evaluation is used, in order to retrieve attribute values, attribute designators and selectors should retrieve values from the partial policy request tuple. For the representation of a retrieved values set that are known partially, an open bag construct is introduced.

Definition 6.14. Open bag is a bag of values that contains all known attribute values from a partial policy request tuple, provided that more values can be added for this attribute when the partial policy request tuple is refined□

Ordinary attribute designators and selectors were extended in order to process partial policy request tuples and return ordinary bags or open bags, depending on the fact if new values can be added for this attribute when the partial policy request tuple is refined.

Attribute designators can retrieve only named attributes from the policy request. One named attribute can contain several values (i.e., it can occur in the policy request several times) and can belong to designated objects, action or time attributes. Action attributes are specified within the partial policy vector and can use placeholders, representing that they are known partially. Based on the policy request generation procedure (see Chapter 5), designated object's attributes are generated based on binary property-literals containing object-terms of the designated objects. There are several possibilities as to how these property-literals can be represented partially in the policy request tuple, which should be taken into account by an attribute designator. First of all, in a partial policy vector, dummy objects can be used as designated objects. Therefore, based on dummy objects definition, existing property-literals for them can be represented partially and new property-literals can be added. Even if these objects are not dummies, their property-literals can be part of high-level effects and, correspondingly, they also can be partial-literals and contain placeholders. Finally, the predicate symbol for a property-literal (it is used as an attribute identifier in the policy request) can be contained in IE (or DE) set and, therefore, more property-literals with this predicate symbol can be added (or some of them can be removed) in the policy request further during its refinement. Time attributes of the policy request are not considered in this

section, their processing is described in Section 6.3.7.

Definition 6.15. Open Attribute Designator has the same behaviour as the attribute designator, but it should return an open bag with attribute values retrieved from the partial policy request tuple, instead of an ordinary bag, in the following cases:

- An attribute of a designated object is being retrieved and this object is a dummy object, that is, its object-term is contained in the set $Term^{DumObj}$. The object-term for the designated object is specified in the policy request as a value of attribute with $ParName = \mathbf{id}$.
- An attribute of a designated object is being retrieved and the identifier of attribute is included in the IE or DE set of the partial policy tuple.
- Among other attribute values retrieved by this attribute designator, a null property, an object-term for a dummy object or a non-leaf hierarchical property is contained[15] $\square$

All these situations indicate that the attribute values retrieved by this attribute designator can be augmented, when the partial policy request tuple is refined. In some special cases, an empty open bag should be returned even if some values were retrieved from the request. An empty open bag is returned when an identifier for designated object is being retrieved (it is contained in **id** attribute) and an object-term for a dummy object is received. In this case, the value retrieved by the designator can be substituted during the further planning, so it is removed from the resulting bag. When the identifier of the attribute being retrieved is contained in the DE set of the policy request tuple, the attribute designator always retrieves an empty bag, since literals with the corresponding predicate symbols are removed from the planner's world state which is used for the partial policy request construction (see Section 6.2.2.2). Hierarchical properties are not removed from an open bag returned by an Open attribute designator, since they are analysed using a specialised procedure during the evaluation of higher-level expression (see Section 6.3.6.3).

The Open attribute designator returns open bags in all situations when new values can be added into resulting bag, when the partial policy request tuple is refined. When a bag of retrieved values can be eliminated, an empty open bag is returned. The approximation order on a disjoint union of all possible bags and open bags representing possible modifications of a retrieved bag during the partial policy request tuple refinement is defined as a subset or equal relation (which takes into account repetitive values also). So when the partial policy request tuple is refined, a bag or open bag retrieved by an Open attribute designator is preserved constant or also refined according to the defined approximation order.

An attribute selector using XPath expression retrieves attribute values from contexts of designated objects placed into the policy request in the form of XML document. Nodes retrieved by the XPath expression are interpreted as a bag of values. According to the object context generation

[15] The null property or the object-term for a dummy-object are not included in the resulting open bag.

technique (see Section 5.6), the object context, constructed for a partial policy request, can contain dummy objects, null and hierarchical properties. In order to determine if more values can be added into a node set retrieved by an attribute selector or some nodes can be substituted, the resulting node set should be analysed. Additionally, since XPath expressions are queries over XML documents that use location paths to navigate to a resulting node set, it is required to trace paths in the object context that can be used to reach resulting node set during the XPath expression evaluation. For this purpose, it is required to analyse location paths that form the XPath expression. Location paths are constructed as a sequence (or a tree-structure) of location steps. Each location step in a location path refers to a set of nodes in the XML document being processed, which are determined relatively to the XML document root node. Only a subset of the XPath specification is supported by the policy-based planner imposing the constraint that each node, used during the XPath evaluation, should be explicitly referred by some location step is the XPath expression[16]. Within location steps specifications in an XPath expression, object types and predicate symbols for properties and relations are used in order to define which nodes these location steps should refer to. So objects which a location step refers to can be dummy objects and more properties can be added for it. Referred relations and properties can be included in the IE or DE sets and, hence, more nodes that the location step refers can be added or they can be eliminated. These situations should be considered in order to determine if an open bag (or an empty open bag) should be returned by the attribute selector.

Definition 6.16. Open Attribute Selector has the same behaviour as the attribute selector, but it should return an open bag with attribute values retrieved from the partial policy request tuple, instead of an ordinal bag, in the following cases:

- Some location step refers to a node set that contains an object-term for a dummy object.
- XPath expression retrieves a null property, an object-term for a dummy object or a non-leaf hierarchical property[17].
- Some location step uses as a node name a predicate symbol from the IE or DE set□

In all these cases more values can be added into the result nodes set when the policy request tuple being evaluated is refined. The second and third conditions can be detected based on the values of nodes retrieved by the attribute selector or using analysis of nodes names, utilised in the XPath expression. In order to decide if the first condition is satisfied, first of all it is checked if in this planning state there is a dummy object with a type mentioned in the XPath expression. If a location step referring to this object type is found, a set of nodes that this location path refers to is determined. For this purpose, a sub-XPath expression is constructed to retrieve a set of nodes

[16]Location paths and specific constraints on supported XPath expressions are considered in more details in Chapter 8.
[17]The null property or the dummy object identifier are not included in the resulting open bag.

that this location step refers to. If a dummy object is included in this object set, the open bag should be returned, since the evaluation of further location steps can be based on properties and relation of this object.

Additionally, an empty open bag should be returned regardless of the number of nodes retrieved if in the predicate part of an XPath expression:

- Some location step refers to a node set containing a dummy object.
- Some location path retrieves a null property, a non-leaf hierarchical property-term or an object-term for a dummy object.
- Some location step uses as a node name a predicate symbol from the IE or DE set.

In these cases an empty open bag is returned, as nodes retrieved by an XPaths expression can be eliminated when the partial policy request tuple is refined. Analysis of a predicate part in an XPath expression is distinct, as it can contain nested location paths that specify conditions restricting a set of nodes referred by the location step, that this predicate part belongs to. Predicate parts can contain different conditions, including negation. So when a number of nodes referred by some location step within a predicate part is extended, this can lead to the reduction of nodes retrieved by the location path that this predicate part belongs to. So when this situation is detected, an empty open bag is returned. Techniques for the analysis of location paths within predicate parts are the same as for other location paths. So the Open attribute selector returns open bags in all situations when new values can be added into the resultant bag. When the bag of retrieved values can be eliminated, an empty open bag is returned.

6.3.6.2 Target evaluation and truth-value functions with Indeterminate Temporal value support

Truth-value functions are used in targets and conditions. For the partial policy evaluation, they are defined as operations on the set $TRVal^p = \{True, False, IndPerm, IndTemp\}$, which contains additionally Indeterminate Temporal truth value $IndTemp$. There are three truth-value operations in the partial policy evaluation, as well as in the ordinary evaluation: conjunction $\wedge^p$, disjunction $\vee^p$ and negation $\neg^p$. Their tables of values are presented in Table 6.5. As can be seen, they were designed as an extension to the truth-value operations used in the standard XACML. Similarly with the standard XACML truth-value operations, the new operations $\vee^p$ and $\wedge^p$ are commutative, idempotent and associative. Additionally, the identity and absorption values True and False are preserved for them. However, in comparison with the standard operations, orders defined based on the new operations are not duals. The reason for this is the fact that these operations should be monotonic according to the approximation order on the set $TRVal^p$ and when both operations are applied to $IndTemp$ and $IndPerm$, the $IndTemp$ value should be returned. Indeed, since $IndTemp$ can be refined into any permanent decision, it should be guaranteed that this refinement

Table 6.5: "$A \wedge^p B$" and "$A \vee^p B$" operations definitions in $TRVal^p$

$A \wedge^p B$	False	IndTemp	IndPerm	True
False	F	F	F	F
IndTemp	F	IT	IT	IT
IndPerm	F	IT	IP	IP
True	F	IT	IP	T

$A \vee^p B$	False	IndPerm	IndTemp	True
False	F	IP	IT	T
IndPerm	IP	IP	IT	T
IndTemp	IT	IT	IT	T
True	T	T	T	T

Table 6.6: "$\neg^p A$" operation definition in $TRVal^p$

	F	IP	IT	T
	T	IP	IT	F

will not lead to updates of operation values that are not refinements. This is possible only when the operation result is always $IndTemp$ if $IndTemp$ is used as an operand, with the exception for the absorbing values. When an absorbing value is used as an operand, regardless of the other operand values, the absorbing value should be returned as the resultant value. As can be checked based on the tables of values, this behaviour guarantees the monotonicity of operations. Correspondingly, the distribution and absorption laws for the conjunction and disjunction operations over the set $TRVal^p$ are not tautologies[18].

The negation operation $\neg^p$ preserves under negation both Indeterminate values: $\neg^p(IndTemp) = IndTemp$ and $\neg^p(IndPerm) = IndPerm$. Obviously, Indeterminate values should not be changed to True or False values under negation. The temporal characteristic of the indeterminate value also cannot be changed, since $IndPerm$ and $IndTemp$ have different but not opposite meanings. For example, it is wrong that "If a is temporally unknown, its negation is unknown permanently".

All functions used during the target evaluation in the ordinary policy evaluation can be represented using conjunction and disjunction (see Section 4.5.5.1). Hence, target evaluation for the partial policy evaluation can be represented similarly using the defined truth-value functions $\vee^p$ and $\wedge^p$. A lower level matcher, which executes a given function over a constant value and a bag of values, requires additional modifications. The bag of values is produced using the Open attribute designator or the Open attribute selector. When an open bag is returned by this component, this bag should be processed using the following mechanism: for each value in this bag it should be checked, if it matches a constant value according to the function provided. If some of the values in the open bag match, the True value should be returned. Otherwise, $IndTemp$ should be produced by this matcher. The monotonicity requirement is satisfied by such a matcher. When one match

[18]Counterexamples are $a \vee (a \wedge b) \neq a$, if $a = IP$ and $b = IT$ for absorbing and $a \vee (b \wedge c) \neq (a \vee b) \wedge (a \vee c)$ if $a = IP$, $b = 0$, $c = IT$ for distribution.

occurs with a value in the open bag, it will occur when this bag is refined, because values from an open bag cannot be eliminated during the refinement. When an empty open bag is returned by the attribute selector or designator, $IndTemp$ should be produced, since this bag can be augmented with new values.

6.3.6.3 Condition evaluation functions

In Chapter 4, the condition is formalised as a composition of functions, constructed according to a specification of condition in a policy rule. For the partial policy evaluation, these functions should be extended to correctly process open bags, retrieved by the Open attribute designators and selectors. During the evaluation of a condition, each of these bags is transformed into a truth value returned as a condition evaluation result. Each transformation is carried out by one function. All possible transformations were analysed and grouped according to the involved abstract data types in Section 4.5.5.2. A chart with all possible transformations was presented in Figure 4.5. In the partial policy evaluation, signatures for all functions that have bag arguments or return bags of values are updated in order to have the possibility to process and return both ordinary bags and open bags. In this section, each of possible transformations is considered and the behaviour of functions, implementing these transformations, is specified for partial policy evaluation. The monotonicity of functions within each class is analysed.

'Bag - Value' and 'Value - Value' transformations.

There are two functions that implement 'Bag - Value' transformation: 'Int-is-one-only', 'Int-bag-size'[19]. The former function gets a bag with one value and returns the individual value, corresponding to it. If the type of input value does not correspond to the output type or if an input bag contains more than one value (or none of them), the Indeterminate result is returned. The latter function returns an integer value equal to a number of values in its Bag argument.

When an open bag is used as an argument of these functions, its output value is undefined: it can become Indeterminate (e.g., if more values are added into a bag processed by 'Int-is-one-only') or it can increase (e.g., if more values are added into a bag processed by 'Int-bag-size'). In order to represent a value that can be substituted by any value with the corresponding data type special temporal zero elements were defined for each data type: Θ^{type} (for integer type it is Θ^{Int}). When this element is added to a set representing all possible values with corresponding data type, a discrete flat order is defined on the resulting set: temporal zero element can be refined into any other value, including Indeterminate value. In the partial policy evaluation, all functions which process or return individual values are defined on lifted sets of possible values with corresponding data types that among other values include a temporal zero element.

When an input bag for the function 'Int-is-one-only' or 'Int-bag-size' is an open bag, it should

[19] In this section, examples of functions consider only integer data type, but in the XACML specification the same functions are defined for other data types.

return a temporal zero element Θ^{Int}. It represents the fact that a value of this function is unknown and can be refined into any value with the corresponding data type, including the Indeterminate value. These functions are monotonic, since when an input open bag is refined into another open bag a value of function is always constant and is equal to the temporal zero. When the input open bag is refined into an ordinary bag, the resultant temporal zero is refined into a concrete value.

When a temporal zero value is processed by a function implementing 'Value - Value' transformation (e.g., arithmetic operations) the temporal zero should be processed in a strict manner: if an input value is a temporal zero, an output value of the function should also be a temporal zero. This guarantees the monotonicity of the function.

'Value - Truth' transformation.

As an example of these functions, comparison and equality functions can be considered (e.g., 'equality', 'more', 'less', 'inequality'). The domains and co-domain for these functions are flat, so in order to guarantee the monotonicity they return Indeterminate Temporal value when at least one of its arguments is a temporal zero value.

'Value - Bag' transformation.

There is only one function implementing this transformation in XACML: 'Int-bag'. This function can be applied to different number of arguments with the same data type. It returns a bag containing all values from the input arguments. A set of functions that represent formally this XACML function should process temporal zero values. When a temporal zero is used as an argument, this function should not include it into the output bag, but it should produce an open bag instead of an ordinary bag. Functions defined according to this principle are monotonic, since when its input argument is refined from a temporal zero into any other value (including Indeterminate value), this value should be added into the output bag. Correspondingly, if the content of the output bag is augmented, a new output bag refines the bag that was returned before.

'Bag - Bag' transformation.

There are three XACML functions implementing this transformation: 'Int-union', 'Int-intersection' and 'map'. 'Int-union' and 'Int-intersection' process two bags and return, respectively, their union and intersection without considering repetitive elements. The map function transforms all values from an input bag into values of an output bag using a function whose name is provided as the function's argument. When one of these functions receives an open bag as an argument, it should return output values in the open bag also. This guarantees the monotonicity, since the addition of new elements into argument bags for these functions will not lead to the elimination of the resulting bag. So when one or several input bags are refined, the output bag is constant or is also refined, if new elements are added.

'Bag - Truth' transformation.

Functions implementing this transformation should be divided into several classes according

to the possible modifications of output truth values that these functions can introduce when new values are added into their input bags. If we consider an additional order $\leq_{tr}$ on the set $TRVal$, such that $False \leq_{tr} Ind \leq_{tr} True$, these classes can be defined based on a monotonicity property of these functions.

Increasing function is a function whose output value can be changed only from False to True according to the order $\leq_{tr}$, when new values are added into any of its bag arguments. Generally, these are functions that require at least one matching between elements of input bags. For example, function 'Int-is-in' processes an integer value and a bag of integer values and returns a True if this value is contained in the bag. Function 'any-of' follows the same principle, but additionally it receives a function name as an argument that should be used to compare values.

Decreasing functions is a function whose output value can be changed only from True to False according to a dual order for $\leq_{tr}$, when new values are added into any of its bag arguments. Generally, these functions require matches for all values within their input bags. For example, function 'Int-set-equals' processes two bags of values and returns true only if these bags are equal (without considering repetitive values). Function 'all-of' is analogous to 'any-of', but it returns true only if the constant input value matches all values in the bag[20].

Left-argument-increasing function is a function that has two bag arguments. If new values are added into its left bag argument, while its right bag argument is constant, the value can be changed only from False to True, according to the order $\leq_{tr}$. If new values are added into its right bag argument, while the left argument is constant, the value can be changed only from True to False, according to a dual order for $\leq_{tr}$. An example of such function is 'any-of-all', which in addition to two bags receives a function name as an argument. This function returns true if any value from the left bag matches all values in the right bag according to the provided function.

Left-argument-decreasing function is a function opposite to Left-argument-increasing function. An example of this function is 'all-of-any' XACML function, which is true if any value from the right bag matches all values in the left bag according to the provided function.

Modifications of these functions required for their usage in the partial policy evaluation obviously depend on the type of a function. An overall mechanism for usage of these functions during the partial policy evaluation is presented in Figure 6.8. Several correction functions were specified that should be applied to a value returned by an original function when one of its arguments is an open bag. Each correction function is a function from the set $TRVal$ to the set $TRVal^p$ that introduces updates required to guarantee the monotonicity of the functions composition. These functions are

[20]During the partial policy evaluation, this and other higher order functions process evaluation results for constituent functions as if these function are connected with conjunction or disjunction operations (see Section 6.3.6.2).

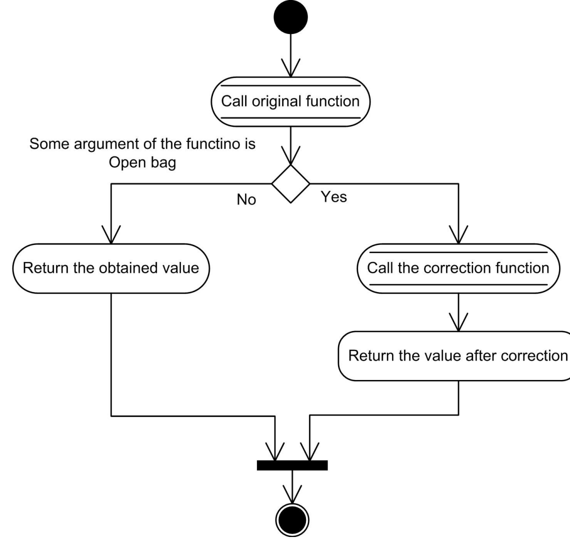

Figure 6.8: Correction mechanism, applied to function '*Func*'

True-preserving correction function γ_{Tr}, **False-preserving correction function** γ_{Fl} and **Constant correction function** γ_{All}. As follows from their tables of values in Table 6.7, the True-preserving correction function modifies the False and Indeterminate Permanent values, the False-preserving correction function modifies the True and Indeterminate Permanent values and maps them to Indeterminate Temporal. Other input values for these functions are preserved. The constant correction function maps all input values to Indeterminate Temporal.

The required correction function should be chosen for each of the considered classed of functions. The correction function should preserve only values that cannot be updated during the refinement of its input open bags. For values that can be updated during the input open bag refinement, Indeterminate Temporal should be returned. For the increasing functions, since True is a supremum for the order $\leq_{tr}$ in $TRVal$, True value cannot be changed during the input open bag refinement. So the True-preserving correction function should be used for their correction: $Func_{Inc} \Rightarrow \gamma_{Tr} \circ Func_{Inc}$. Based on the same principle, decreasing functions should be corrected by the False-preserving correction function since False is an infinum for the order $\leq_{tr}$ in $TRVal$: $Func_{Decr} \Rightarrow \gamma_{Fl} \circ Func_{Decr}$. For the Left- and Right-argument-increasing functions, the choice of a correction function depends on the fact in which argument an open bag is used. For the Left-argument-increasing function when its left argument is an open bag and right argument is an ordinary bag, True-preserving correction function should be used. When the left argument is an ordinary bag and the right argument is an open bag, the False-preserving correction function should be used. When both arguments are open bags, there is no guarantee if the output value will increase or decrease. Hence, the Constant correction function should be used: $Func_{Const} \Rightarrow \gamma_{All} \circ Func_{Const}$. For the Left-argument-decreasing function, the True- and False-preserving functions are utilised

Table 6.7: Tables of values for True- and False-preserving correction functions γ_{Tr} and γ_{Fl} and Constant correction function γ_{All}

A	$\gamma_{Tr}(A)$
True	True
IndPerm	IndTemp
False	IndTemp

A	$\gamma_{Fl}(A)$
True	IndTemp
IndPerm	IndTemp
False	False

A	$\gamma_{All}(A)$
False	IndTemp
IndPerm	IndTemp
False	IndTemp

in the opposite cases. So when an open bag is used as one of the arguments for each of these functions, a composition of functions that are used instead of the original function is guaranteed to be monotonic. Only values that cannot be updated during the possible refinements of input values, are returned as a result of the composition. All values that can be updated during the input values refinements are substituted by the $IndTemp$ value, which can be refined into any other value.

'Truth - Truth' transformation.

Functions implementing this transformation are the same with the truth-value functions considered in Section 6.3.6.2.

Additionally, during the condition evaluation, non-leaf hierarchical properties (see Section 6.2.1) retrieved by an attribute designator or selector can be a processed. These hierarchical properties will be refined into lower level hierarchical properties and this should be considered during the evaluation of expressions. When a hierarchical property which is not a leaf node in the corresponding hierarchy is processed by a function, distinct from the equality relation, it should return a temporal zero value or $IndTemp$, because its specific value is not known. The equality function processes hierarchical properties based on their positions in the corresponding hierarchy of properties. If these values belong to different hierarchies, $False$ is returned. Otherwise, for their processing, a hierarchical property $\tau_i^{\mathcal{G}}$ is associated with a set of its descendant leaf nodes: $design(\tau_i^{\mathcal{G}}) = \{\tau_{i1}^{\mathcal{G}}, \ldots, \tau_{in}^{\mathcal{G}}\}$ and these sets are used to compare positions of different properties in the hierarchy. Additionally, it is distinguished if the hierarchical property was retrieved from the policy request (in this case, it can be refined further) or it is contained in a policy as a constant value. Constant values are denoted as $\tau_{iconst}^{\mathcal{G}}$. When sets of leaf nodes corresponding to two arguments of the equality function do not intersect: $design(\tau_i^{\mathcal{G}}) \cap design(\tau_j^{\mathcal{G}}) = \emptyset$ (regardless if they are constant or not), they cannot represent the same value and False is returned. If both arguments are not constants, True is returned if these hierarchical properties are equal and both are leaf-nodes, otherwise $IndTemp$ is produced. Two constant values are processed as ordinal values and True is returned if they are equal, False is produced otherwise. If only one of the arguments is constant, True is returned when a set of leaf nodes for the non-constant property is included or equal to a leaf nodes set for another operand. In this case, a value that the non-constant hierarchical property will be refined into is guaranteed to be included in the set of values represented by the constant value. Otherwise, the concrete decision is undefined and $IndTemp$ is returned.

6.3.7 Loose time intervals processing

During the evaluation of policies, loose time intervals $TInterval^p = \langle ActBeg, ActEnd \rangle$ are processed using the same procedure as ordinary time intervals (see Section 5.4.2.2). So a policy is considered applicable, if it does not have an operation period (i.e., it is always applicable) or if its operation period intersects with the time interval specified in the policy request, which can be a loose time interval. Additionally, when the loose time interval is considered, decisions produced by policies applicable only during some subset of this time interval are modified, since when it will be substituted by a strict time interval such policies can become inapplicable. So if these policies are applicable to the current situation (the decision produced is not N/A), the $IndTemp$ decision is returned regardless of the decision produced by the core part of the policy. The $IndTemp$ decision represents the fact that the decision can be updated when the strict time interval will be defined.

6.4 Conclusion

The main contribution of this chapter is the postponed policy enforcement mechanism that was introduced to improve the planning performance for the policy-based planner, designed in Chapter 5. The principle that earlier recognition of dead-ends during the planning can improve the planning performance was applied to the policy enforcement process in the policy-based planner. In the postponed policy enforcement, policies can be evaluated at earlier stages of the planning. For this purpose, partial policy requests corresponding to actions that should be executed during the future course of the planning are generated and evaluated. Decisions that can be produced based on these requests are enforced during the planning. If this decision is Deny, a dead-end is detected and a large part of the search space can be pruned. If this decision is Permit, the evaluation of future policy requests that refine the current request can be eliminated. This leads to the planning time reduction and provides the means to produce the solution faster.

Another contribution of this chapter is the extension of the standard XACML policy evaluation and the introduction of the partial policy evaluation procedure supporting the partial policy requests evaluation. During the generation of policy requests in the postponed policy enforcement, not all required information could be available. Correspondingly, a new mechanism was designed in order to construct partial policy requests containing only known part of the information about the future policy request along with the indications on modifications that can be expected. For the evaluation of these partial policy requests, the standard XACML policy evaluation mechanism was extended and the partial policy evaluation procedure was introduced. In order to represent a situation when a standard XACML policy decision cannot be inferred during the evaluation, a new Indeterminate Temporal decision was introduced. When an Indeterminate Temporal decision is produced during the planning, the partial policy request should be postponed and re-evaluated when more information is available. The partial policy evaluation procedure was designed as an

extension of the formal model of the XACML policy evaluation introduced in Chapter 4. Using this formal model, it is possible to guarantee that the partial policy evaluation possesses properties required for its correct usage during the postponed policy enforcement: the partial policy evaluation should be an extension of the ordinary policy evaluation and should be monotonic, meaning that, when a partial policy request is refined during the planning, the policy decision should be preserved constant or also refined according to the specified approximation order.

The postponed policy enforcement mechanism will be utilised when the planning domain for the CEP generation problem is specified (see Chapter 7). This mechanism will be used for the development of a descending policy evaluation technique. The descending policy evaluation is a problem-specific technique that optimises the process of EPs selection during the CEP development and utilises the postponed policy enforcement as its basic principle.

Chapter 7

Planning for Combined Educational Programmes development

Objectives:

- *Describe specifications designed to tailor the policy-based planner for solving the CEP generation problem, including the planning domain specification.*
- *Describe the descending policy evaluation technique that optimises the EPs selection process during the CEP development and is aimed at the planning performance improvement.*

7.1 Introduction

In order to solve the CEP generation problem using the policy-based planner, designed in Chapter 5 as a problem-independent planning engine, it should be provided with a specially designed planning environment specification and the CEP generation problem should be formalised as a planning problem in this environment. This chapter presents the specification of the planning environment for the CEP development, including the models of Learning objects (LObj) utilised for the CEP construction, and describes how the CEP generation problem is formalised and solved in this environment.

In Section 7.2, all specifications that refer to LObjs and their relations are covered. In Subsection 7.2.1, we describe how different LObjs utilised in the CEP generation framework are specified. These LObjs include existing objects provided for the CEP generation as input and new objects generated by the solution for the internal use or as a CEP generation outcome. In Subsection 7.2.2, we present different measures, which can be used to compare the content of LObjs, developed in different universities. In Subsection 7.2.3, we specify a hierarchical multi-domain environment, which defines the overall structure of the planning environment for the CEP development. Domains represent different areas or entities within the HE environment, organised hierarchically.

This model of the environment relate LObjs designed in different domains and policies specified for these LObjs at different levels of the educational system. Moreover, the hierarchical structure of the environment provides the possibility to carry out planning at different levels of this hierarchy. Finally, in Subsection 7.2.4, we describe how properties of LObjs that can be described using terms and units adopted in different classification systems and scales (i.e., educational levels and credits) are specified and converted using the designed transformation rules.

In Section 7.3, all specifications and algorithms related to the core planning processes are covered. In Subsection 7.3.1, we describe how CEP requirements can be specified by a user[1]. These requirements form the planning problem statement. Additionally, in Subsection 7.3.1, the designed planning domain specification, viz., operators and methods that are used to actually carry out planning for the CEP development, is described. In Subsection 7.3.2, we describe the descending policy evaluation technique, which was developed to extend the policy-based planning algorithm in order to improve the planning efficiency for the higher-level phase of the CEP construction procedure and provide the means to reduce time required to produce first planning outcomes for the user. The descending policy evaluation technique is based on the utilisation of problem specific characteristics of the CEP generation problem, namely, its hierarchical multi-domain environment, and it is based on the postponed policy enforcement mechanism, which was described in Chapter 6.

7.2 Learning objects and their relations specification

7.2.1 Learning objects specification

The most general definition of Learning object (LObj), given in IEEE LOM standard, states that "a learning object is defined as any entity - digital or non-digital - that may be used for learning, education or training" [97], p.1. Other approaches to defining LObjs [128, 176, 115] restrict this definition and specify the following LObj properties: LObj is self-contained (LObj can be taken independently from other LObjs), reusable (LObj can be used in multiple educational contexts different from the original one) and has independent educational purpose(s) explicitly stated. We adopt these properties of LObjs for the CEP generation framework. Additionally, in the CEP generation framework, we require that LObjs are described from three perspectives, in accordance with the BP initiatives (see Chapter 3): content, namely, using learning outcomes; workload, namely, using credits; and complexity and depth of study, for this purpose mappings to corresponding levels in educational frameworks are used.

Similarly to other approaches for the LObjs management [128, 176, 115], we assume that LObjs can be aggregated and as a result of their aggregation new LObjs are produced. An atomic LObj in the CEP generation framework is a module.

[1] As was specified in Chapter3, users of the CEP generation solution can be students or members of institutions who develop CEPs for students.

Definition 7.1. Module is defined as a tuple:

$$Mod = \langle Mod_{Name}, Con_{lo}^{S}, Cr, Level, Pre, Prov_{mod} \rangle \tag{7.1}$$

where Mod_{Name} is the title of the module, $Con_{lo}^{S} = \{lo_1, lo_2, ..., lo_n\}$ is the set of learning outcomes of the module, Cr is a specification of the credits that student earns, $Level$ is a specification of the module's level that defines the depth and complexity of the module, Pre is a specification of the module's prerequisites that a student should satisfy in order to study this module, $Prov_{mod}$ is the official education provider for this module□

The representation and comparison of learning outcomes are described in Section 7.2.2. Credits are specified by the tuple $Cr = \langle Cr_{Val}, Cr_{Scale} \rangle$, where Cr_{Val} is the number of credits and Cr_{Scale} is the scale according to which one credit unit is defined. Usually, a country (or other domain) where this credit unit is adopted is used as the scale. The educational level of the module is defined as $Level = \langle Lev_{Name}, Lev_{Scale} \rangle$, where Lev_{Name} is the level name which this module corresponds to and Lev_{Scale} is the qualification framework where this level is defined. Prerequisites $Pre = \langle \{lo_1, lo_2, ..., lo_m\}, \{Mod_1, Mod_2, ..., Mod_k\} \rangle$ contain learning outcomes that the student must gain and modules that he (or she) must study. The module's education provider $Prov_{mod}$ is specified as a university or other lower-level entity within the university structure, for example, a faculty or a school (see Section 7.2.3). Modules can be united into modules groups.

Definition 7.2. Modules group is a subset of modules studied within the same semester. The modules group joins optional modules that are managed by a common set of modules selection rules and is specified as a tuple:

$$Group = \langle Group_{Mod}^{S}, Cr \rangle \tag{7.2}$$

where $Group_{Mod}^{S} = \{Mod_1, Mod_2, ..., Mod_n\}$ is the set of optional modules in this group, Cr is the minimal number of credits that the student should receive for studying this group□

Selection rules that are used to guide the student's choice of modules from this group are specified using the modules selection policy. This policy is used during the planning to manage the optional modules selection process. When a module within modules group is considered for the selection, this policy can permit or deny the corresponding action. Additionally, this policy determines when a modules group can be closed, that is, when enough modules were chosen for the student.

On a higher level of detail, modules and groups of modules are united into semesters.

Definition 7.3. Semester is a LObj that is formed by two sets: a set of modules $Sem_{Mod}^{S} = \{Mod_1, Mod_2, ..., Mod_k\}$ that are compulsory for studying in this semester and a set of modules

groups $Sem^S_{Group} = \{Group_1, Group_2, ..., Group_j\}$ that contain optional modules:

$$Sem = \langle Sem^S_{Mod}, Sem^S_{Group}, Prov_{Sem}, TEMP_{Sem} \rangle \tag{7.3}$$

Additionally, semester contains $Prov_{sem}$ parameter, that is the official education provider where students are studying during this semester (e.g., a university), and $TEMP_{Sem}$ tuple, containing temporal parameters of the semester□

We assume that students can transfer from one EP to another only between semesters, as this happens in the majority of the cases in HE, and that all modules within one semester Sem should be provided by one education provider $Prov_{sem}$[2]. The tuple $TEMP_{Sem} = \langle Dur, St, Fin \rangle$ contains the duration of this semester in days Dur and sequence numbers of months of the year when this semester starts and finishes: St and Fin[3]. We assume that all modules included into a semester are studied in parallel, so they all share the same temporal properties of the semester.

Definition 7.4. Educational programme (EP) is a LObj formed by a sequence of semesters at the end of which one or several awards are granted:

$$EP = \langle EP^{OS}_{Sem}, EP^S_{AWARD}, Prov_{EP} \rangle \tag{7.4}$$

where $EP^{OS}_{Sem} = \langle Sem_1, Sem_2, \ldots, Sem_p \rangle$ is a tuple of semesters that forms this EP. $EP^S_{AWARD} = \{\ldots Award_i \ldots\}$ is a set of awards that the student gets when he (or she) finishes this programme, $Prov_{EP}$ is the education provider for this EP (e.g., a university)□

Each award $Award_i$ is defined as a tuple $\langle Award_{Name}, Prov_{aw}, AW^S_{Area}, Level \rangle$, where $Award_{Name}$ is the title of the award, $Prov_{aw}$ is the education provider (e.g., a university) that issues this award[4], $AW^S_{Area} = \{Area_1, Area_2, ..., Area_u\}$ is the set of fields of study of this award (usually, there is only one major field, but some awards can include subsidiary fields also) and the tuple $Level = \langle Lev_{Name}, Lev_{Scale} \rangle$ specifies that the level of this award is Lev_{Name} according to the qualifications framework Lev_{Scale}. Usually, the qualifications framework corresponds to a country (or other domain) where the award is issued. The tuple of semesters EP^{OS}_{Sem} contains semesters in the order according to which they will be studied by the student. The semesters are specified in an uninterrupted manner: durations of semesters include durations of adjacent vacations, so, for adjacent semesters, the next semester should start immediately after the end point of the previous semester. EPs defined in such way represent several paths that students can take by selecting dif-

[2]We consider only the pure credits mobility CEP type and do not consider the virtual mobility, which allows to study modules from different providers in parallel (see Chapter 3).

[3]For example, for January '1' is used, for February - '2', etc. Semester duration 'Dur' is an auxiliary parameter since it is considered that semesters start on the first day of the month and end on the last day (the duration of semester cannot be longer than a year). 'Dur' parameter is used to calculate duration of the education easier, when several semesters are studied sequentially.

[4]In our model, one award can be granted only by one university. In order to take into account joint degrees, this part of the model should be extended.

ferent optional modules. When a student studies according to an EP, a concrete educational track is constructed for him (or her): optional modules contained in the modules groups within this EP are selected. The groups themselves should be closed, what indicates that the required number of optional modules was chosen. When an optional module is chosen or a group is closed, a special label is added to it (a special literal with this module or group in the terms list is added into the planner's world state). In our definition, EP is an ordinary educational programme that is provided by one education provider (e.g., by one university). Therefore, all modules and semesters within one EP should be provided by the same education provider $Prov_{EP}$. Education providers for the awards $Prov_{aw}$ can be equal to the EP education provider $Prov_{EP}$ or they can be specified as higher-level entities within the university structure, such that the EP education provider $Prov_{EP}$ is situated within the award education provider $Prov_{aw}$ (for example, EP education provider is a faculty within the university, but awards are issued by the university itself).

In order to define a CEP, we introduce an artificial LObj type: an EP interval. The EP interval is a LObj produced based on an EP by selecting its part such that it can be studied by a student in an uninterrupted way. The EP interval does not contain the EP's award.

Definition 7.5. EP interval from n to m $|EP|_{[n,m]}$ is a tuple of semesters that contains all semesters in EP^{OS}_{Sem} tuple from the nth semester to the mth semester inclusive□[5]

Definition 7.6. Combined educational programme (CEP) is a tuple: $CEP = \langle CEP^{OS}_{intEP}, CEP^{S}_{AWARD} \rangle$, where $CEP^{OS}_{intEP} = \langle |EP_1|_{[n_1,m_1]}, \ldots, |EP_k|_{[n_k,m_k]} \rangle$ is a totally ordered set of EP intervals that form this CEP, CEP^{S}_{AWARD} is the set of awards that the student gets when he (or she) graduates from the CEP□

The order of EP intervals in the set CEP^{OS}_{intEP} corresponds to the order according to which these EP intervals will be studied by the student. An EP interval $|EP_i|_{[n_i,m_i]}$ used at i^{th} position in the tuple CEP^{OS}_{intEP} of a CEP is called an ith slot of this CEP. EP intervals for the CEP can be taken from different universities (education providers), then it is external mobility programme, or they can be from the same university (education provider), then it is internal mobility programme. Some EP intervals used in a CEP can be parts of the same EP, then this CEP implements a probation period mobility scenario (or a temporal mobility scenario, as the student returns to its original EP). EP intervals in a CEP should not overlap in time, but between adjacent semesters some 'idle time' can exist[6].

LObjs that should be accessed during the planning should be placed into the planner's world state. For this purpose, they are represented using literals. For example, the fact that a module

[5]In order to easily retrieve durations of EP intervals during the planning, durations of all possible EP intervals that can be produced based on EPs within the planner's world state are stored within the state using literals '*duration_interval*'. In term lists of these literals, the EP interval is identified using the EP identifier and numbers of start and end semesters. The duration of EP interval is specified in days and, additionally, as number of years and months.

[6]Constraints on the maximum 'idle time' are specified as part of the CEP requirements (see Section 7.3.1.1)

Mod_i has credits tuple $Cr_i = \langle Cr_{Vali}, Cr_{Scalei} \rangle$ is represented as literal $credits_spec(Mod_i, Cr_{Vali}, Cr_{Scalei})$. In the defined LObjs model, not all LObj attributes are specified on each level of the model, that is, not for all types of LObjs. For example, learning outcomes were defined only at modules level, credits - at modules and modules groups levels. Values of attributes corresponding to higher level LObjs, which can be used during the policy evaluation, can be derived as an aggregation of corresponding attributes for its constituent LObjs. As all properties of a LObj and its relations with the constituent LObjs are stored in the planner's world state, any required information about a LObj can be retrieved during the policy evaluation using *AttributeSelector*s, including properties of LObjs that form a higher-level LObj, and represented as a bag of values. Moreover, the policy author can specify restrictions that define which specific values should be retrieved, for example, credits of all compulsory modules in an EP interval, or credits of all selected optional modules within a modules group. So, using the XACML conditions mechanism supporting complex sets-based and numeric-based expressions, it is possible to define any possible aggregations of retrieved values and impose required constraints on them.

7.2.2 Learning outcomes-based relations between Learning objects

Comparison of educational content taught within LObjs is carried out based on their learning outcomes. As the basis for this comparison, we use similarity measures between two individual learning outcomes $sim^{lo}(lo_1, lo_2) \in [0, 1]$ and between two modules $sim^{mod}(Mod_1, Mod_2) \in [0, 1]$, adopted from [32]. These similarity measures show how similar two los or two modules Mods are, based on their textual description[7]. During the CEP development, content of LObjs should be compared when modules prerequisites are evaluated, for the comparison of lo-prerequisites and modules-prerequisites with los and modules that the student has studied, and when a student makes a transfer, for the comparison of modules that the student has studied with the modules in the EP that he (or she) transfers to, in order to make the recognition. Specific constraints that should be satisfied in order fulfil a prerequisite or in order to recognise a module are specified as constraints in corresponding policies. These constraints, among other conditions, should include restrictions on the minimum values of the corresponding similarity measures. For example, the following policy can be specified:"*In order to fulfil a module prerequisite specified as* lo_i *within the module tuple* Mod_i*, similarity measure* $sim(lo_i, lo_j)$ *between* lo_i *and some learning outcome* lo_j *that the student has already achieved should be more then* $N\%$ *threshold value*". Additionally, other constraints can be imposed in such policies, for example, that the learning outcome lo_j should be studied not more then K months ago. Analogous policy constraints can be specified for the fulfilment of prerequisites specified as modules and on the recognition of modules.

[7] In order to obtain values of these similarity measures for specific los and Mods, their textual description should be converted into ontologies with the pre-defined structure using a dedicated recogniser. These ontologies are processed using the alignment algorithm that derives the value of the similarity measure. Details can be found in [32].

Policy conditions containing constraints on the described similarity measures are specified by the policy authors at their discretion and can include arbitrary complex expressions. In order to reduce the complexity of specification of these policy conditions, pre-configured policy functions that are used to derive higher-level similarity measures between learning outcomes, based on values of lower level similarity measures $sim^{lo}(lo_i, lo_j)$ and $sim^{lo}(Mod_i, Mod_j)$, can be introduced. When these functions are used in a policy condition, they should substitute some part of the condition that otherwise has to be specified by the policy author by hand. The following functions were introduced in the CEP development solution being developed, but other functions can also be specified when this is demanded. The first example of such aggregated measure is a maximum similarity measure calculated between a learning outcome lo_j and a learning object $LObj$:

$$\mu_{lo}^{max}(LObj, lo_j) = max_{lo_i \in LObj}\{sim^{lo}(lo_i, lo_j)\} \tag{7.5}$$

where $LObj$ can be a module, a semester or an EP interval. lo_i is a learning outcome in the learning outcomes set Con_{lo}^{S} corresponding to some module in $LObj$ (or to $LObj$ itself if it is a module). This measure shows which the most similar learning outcome exists in $LObj$ for learning outcome lo_j. This measure can be required when lo prerequisite is being evaluated. Then, lo_j is a prerequisite and $LObj$ is an EP interval that the student has studied. The analogous maximum similarity measure for modules $\mu_{mod}^{max}(LObj, Mod_j)$ was implemented based on the $sim^{mod}(Mod_i, Mod_j)$ similarity measure.

Another aggregated measure that was introduced is the average between maximum similarity measures. This measure defined for modules similarity measures is represented in Formula 7.6. The measure $K_{mod}^{avg}(LObj_i, LObj_j)$ is equal to a sum of maximum similarity measure values (μ_{mod}^{max}) calculated between modules contained in $LObj_j$ and the learning object $LObj_i$ divided by the number of modules in $LObj_j$. The $LObj$ can be a semester, an EP or its interval. This measure can be used in order to specify policy conditions based on the fact how at the average modules contained in $LObj_j$ are similar with modules in $LObj_i$. The analogous measure K_{lo}^{avg} can be defined between two LObjs based on the learning outcomes similarity measure[8].

$$K_{mod}^{avg}(LObj_i, LObj_j) = \frac{\sum_{\{Mod_o \in LObj_j\}} \mu_{mod}^{max}(LObj_i, Mod_o)}{|LObj_j|} \tag{7.6}$$

7.2.3 Hierarchical multi-domain structure and policies for Learning objects

The educational environment, where LObjs for the CEP development are stored, has a hierarchical structure. It consists of nested domains where each higher level domain contains objects within

[8]Generally, measures K^{avg} are relations between two sets (either containing learning outcomes, or modules) that are defined similarly with inclusion measures in [139]. The inclusion measure of a set A into a set B indicates which part of the set A is included into the set B, i.e., what is the percentage of elements in A contained also in B.

lower level nested domains. This environment is formalised as a multi-domain hierarchical structure, which is a special type of hierarchy of properties (see Chapter 6). This hierarchy of properties is not only used to specify object properties at different levels of hierarchy, but also it represents an overall structure of the planning environment considered and is connected with policies defined for this planning environment.

Definition 7.7. Multi-domain hierarchical structure (a domain tree) is a hierarchy of properties $\mathcal{D} = \langle Term^{\mathcal{D}}, E^{\mathcal{D}}, \tau_0^{\mathcal{D}} \rangle$, where $Term^{\mathcal{D}} = \{\dots, D_x^y, \dots\} \cup \{\dots, LObj_i, \dots\}$ is the union of all domains D_x^y (they can be countries, universities, regions, etc.) and all LObjs $LObj_i$ stored in the environment. They are organised hierarchically according to $E^{\mathcal{D}} = \{\dots, \langle D_{x_1}^{y_1}, D_{x_2}^{y_2} \rangle, \dots\}$, which defines the structure of the planning environment□

Each domain D_x^y, except the root domain of the tree $\tau_0^{\mathcal{D}}$, has indexes that identify its place in the domain tree: x - level of the domain in a tree, y - number of the domain on this level. Domains represent different entities in the educational environment on a higher-level, for example, countries, regions, universities, and entities within the university, for example, faculties, schools, which provide the educational services. Education providers for modules, semesters and EPs $Prov_{Mod}$, $Prov_{Sem}$ and $Prov_{EP}$ are specified as smallest, most specific domains within the domain tree (i.e., as domains that do not have children). So LObjs are added into the domain tree as leaf-nodes, which are children of their education provider domains $Prov_{LObj}$. CEPs are not included in these $LObj$s, since they are produced during the planning and are not stored in the environment. Therefore, each domain in a domain tree structure is defined as set of its descendant LObjs: $D_x^y = \{\dots, LObj_i, \dots\}$. The fact that the domain $D_{x_i}^{y_i}$ is a descendant of the domain $D_{x_j}^{y_j}$ or is equal to it can be designated as $D_{x_i}^{y_i} \subseteq D_{x_j}^{y_j}$. Edges of the domain tree are specified using predicates $in(D_i, D_j)$, where D_j is a child domain or LObj and D_i is a parent domain.

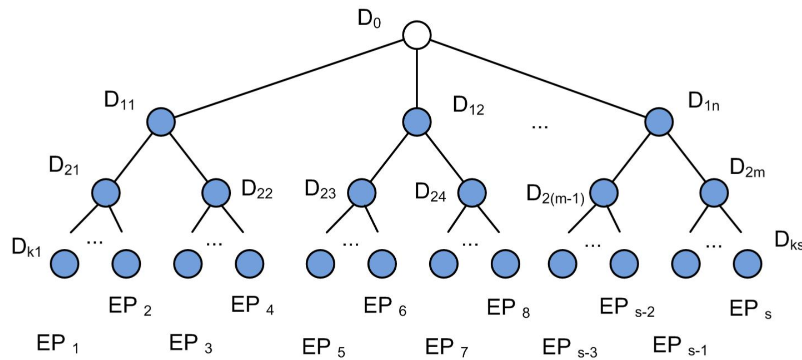

Figure 7.1: Hierarchical multi-domain environment (domain tree)

In the CEP generation problem, policies have hierarchical structure reflecting the domain tree of the environment. Each domain D_x^y and learning object $LObj_i$ in the domain tree corresponds to a policy set Pol_x^y (for some domains and LObjs, the policy set can be empty). The policy

set Pol_x^y corresponding to the domain D_x^y is applicable only to LObjs situated within D_x^y. The policy set $Pol_{x'}^{y'}$ corresponding to a LObj is applicable to this LObjs and all its constituent LObjs. This policy set contains policies managing different aspects of the educational processes and LObjs usage within this domain (within this policy set different policy organisations are possible, which are determined by local practises). Since domains are nested in a hierarchical structure, scopes of policies corresponding to domains connected with 'ascendent-descendent' relation intersect. When these policies generate conflicting decisions, a deny-overrides policy combining algorithm is used. So a policy request should be permitted only when the policy set of the domain where the action is carried out is permitted and all policy sets corresponding to its ancestor domains are permitted (or some of these policy sets can be evaluated to 'N/A'). An example of policy structure illustrating organisation of higher-level policies in a hierarchical multi-domain environment is presented in Formula 7.7. In this example, domain D_0 is the root domain containing domains D_1^1 and D_1^2. Each of these domains contains two sub-domains: D_2^1, D_2^2 and D_3^1, D_4^2. In addition to the policies shown in the formula, these domains can contain their own policies. Then, these policies should be added into the corresponding policy sets as additional operands of the combining operations.

$$\begin{aligned}
&P^e(evaluate^T([\![\texttt{Target}_{D_0}]\!], req),\\
&\quad P^e(evaluate^T([\![\texttt{Target}_{D_1^1}]\!], req),\\
&\quad evaluate^{PS}([\![\texttt{PolicySet}_{D_2^1}]\!], req) \bullet_p^{DO} evaluate^{PS}([\![\texttt{PolicySet}_{D_2^2}]\!], req), req)\\
&\bullet_p^{DO}\\
&\quad P^e(evaluate^T([\![\texttt{Target}_{D_1^2}]\!], req),\\
&\quad evaluate^{PS}([\![\texttt{PolicySet}_{D_2^3}]\!], req) \bullet_p^{DO} evaluate^{PS}([\![\texttt{PolicySet}_{D_2^4}]\!], req), req), req)
\end{aligned} \tag{7.7}$$

In the targets of policies that form the overall policy hierarchy $\texttt{Target}_{D_1^1}$ and $\texttt{Target}_{D_1^2}$ and targets of policy sets $\texttt{PolicySet}_{D_2^1}$, $\texttt{PolicySet}_{D_2^2}$, $\texttt{PolicySet}_{D_2^3}$ and $\texttt{PolicySet}_{D_2^4}$, constraints on the set of LObjs that these policies are applicable to should be specified. In actions used in the planning domain specified (see Section 7.3.1), the main LObj that the action is applied to is contained with the role '*resource*'. This LObj determines the domain where this action is carried out. So, in order to form the hierarchy, constraints in these targets should be satisfied only when a LObj used in the policy request as '*resource*' is within the domain that these policy sets correspond to. As a domain tree is a hierarchy of properties, this condition can be represented using an equality function, which for hierarchical properties is evaluated according to the 'equal or included' principle (see Section 6.3.6.3).

7.2.4 Transformation rules for Learning objects properties

In the policy-based planner, transformation rules are used to convert properties of objects specified using terms adopted in one domain or classification system to another (viz., from one scale to another). Numeric properties are converted using conversion expressions. For non-numeric values, mappings of values from one scale to another are defined. Generally, notions from different scales can be related directly or through an intermediate scale.

For credit values conversion, two ways are used. When it is required to convert a credit value in a national credit system into another national system, an intermediate representation using literal $hours(LObj, H_{LObj})$ is used. It indicates how many notional hours is required to study a learning objects $LObj$. An example of transformation rules for conversion of credit values from one national credit system ($ScaleA$) to another national credit system ($ScaleB$) is presented in Formula 7.8. Using the first rule, it is possible to derive how many notional hours H_{LObj} it takes to study a LObj, for which credits values are specified in $ScaleA$. Using the conversion expression $?H_{LObj} = ?Cr_{ValA} \cdot Rate_A$, where $Rate_A$ is the number of notional hours corresponding to one credit in $ScaleA$, the core calculations are carried out. The second rule defines how many credits should be allocated to LObj in the national credits system $ScaleB$. Predicate symbol *credits* is utilised in policies and the planning domain specification when it is required to refer to credits values of LObjs. Using this predicate, values converted according to the described procedure are accessed, as well it is possible to retrieve a credit value of LObj in the national credit system as it is specified in the planner's world state (for this purpose, the third rule in Formula 7.8 is used). For conversion of credits from $ScaleB$ to $ScaleA$, transformation rules analogous to rules 1 and 2 in Formula 7.8 should be specified.

$$\begin{gathered} credits_spec(?LObj, ?Cr_{ValA}, Cr_{ScaleA}), (?H_{LObj} = ?Cr_{ValA} \cdot Rate_A) \rightarrow hours(?LObj, ?H_{LObj}) \\ hours(?LObj, ?H_{LObj}), (?Cr_{ValB} = ?H_{LObj}/Rate_B) \rightarrow credits(?LObj, ?Cr_{ValB}, Cr_{ScaleB}) \\ credits_spec(?LObj, ?Cr_{Val}, ?Cr_{Scale}) \rightarrow credits(?LObj, ?Cr_{Val}, ?Cr_{Scale}) \end{gathered} \tag{7.8}$$

On the other hand, credit values are converted without intermediate representation in notional hours when it is required to convert national credits into ECTS credits: $credits_spec(?LObj, ?Cr_{ValN}, ?Cr_{ScaleN}), (?Cr_{Val} = expr(?Cr_{ValN})) \rightarrow credits(?LObj, ?Cr_{Val}, ECTS)$, where $expr()$ is an expression for conversion of national credits $?Cr_{ValN}$ into ECTS credits $?Cr_{Val}$, which can be specified using more complex expressions than a linear function.

Qualification levels in NQFs are mapped to levels EHEA QF. Mappings between specific levels in NQFs and levels of EHEA QF are stored in the planner's world state using literals $equiv(Lev^{EHEA}_{Name}, Lev^{NQF}_{Name}, Lev^{NQF}_{Scale})$, where Lev^{NQF}_{Name} is a level in NQF Lev^{NQF}_{Scale} being equiva-

lent to level Lev_{Name}^{EHEA} in EHEA QF. Using these mappings, based on an educational level of an award in NQF it is possible to determine which level in EHEA QF this award corresponds to. For this purpose, the rule in Formula 7.9 is used. Literals $level(Award, Lev_{Name}, Lev_{Scale})$ are used to denote a fact that an award corresponds to a certain level Lev_{Name} in the framework Lev_{Scale}, which can be either a NQF or an EHEA QF. In order to retrieve all awards corresponding to the same EHEA QF level, literal $eqLev(Award, Lev_{Name}, Lev_{Scale})$ is used (see Formulae 7.10, 7.11). In the rule in Formula 7.10, an EHEA QF level that these awards should correspond is specified explicitly in the *eqLev* literal, as Lev_{Name}. In the rule in Formula 7.11, instead of it a NQF level is specified and the awards should correspond to an EHEA QF level being equivalent to this level.

$$level(?Award, ?Lev_{Name}^{NQF}, ?Lev_{Scale}^{NQF}), equiv(?Lev_{Name}^{EHEA}, ?Lev_{Name}^{NQF}, ?Lev_{Scale}^{NQF}) \rightarrow \\ \rightarrow level(?Award, ?Lev_{Name}^{EHEA}, ehea_qf) \quad (7.9)$$

$$level(?Award, ?Lev_{Name}^{EHEA}, ehea_qf) \rightarrow eqLev(?Award, ?Lev_{Name}^{EHEA}, ehea_qf) \quad (7.10)$$

$$level(?Award, ?Lev_{Name}^{EHEA}, ehea_qf), equiv(?Lev_{Name}^{EHEA}, ?Lev_{Name}^{NQF}, ?Lev_{Scale}^{NQF}) \rightarrow \\ \rightarrow eqLev(?Award, ?Lev_{Name}^{NQF}, ?Lev_{Scale}^{NQF}) \quad (7.11)$$

Using transformation rules, it is also possible to define correspondence between other properties, for example, language qualifications or fields of study for awards. Currently, only one scale is supported for each of these properties: respectively, the Common European Framework of Reference for Languages (CEFR) [36] and the International Standard Classification of Education (ISCED)'s fields of education classification [166].

7.3 Planning procedures for CEP development

7.3.1 HTN planning domain for CEP generation process

7.3.1.1 Input requirements for CEP generation

The input requirements for the CEP generation are specified as a tuple $\langle Req^{Struc}, Req^{Prop}, Req^{Proc}, Student, \mathcal{D}\rangle$, where Req^{Struc}, Req^{Prop} and Req^{Proc} are requirements for the CEP from the structural, properties and process perspectives respectively, *Student* is the student for whom the CEP should be developed and $\mathcal{D}$ is the domain tree that specifies the educational environment for the CEP development.

The *structural requirements* Req^{Struc} are specified as an Initial track: $Req^{Struc} = ITr$. The definition of the track is based on the notion of the CEP's slot (it is an EP interval used for the CEP construction at the specific position of the CEP structure, see Section 7.2.1).

Definition 7.8. Track is a sequence of domains, EPs and EP intervals that defines the structure of CEP. It divides the CEP into a sequence of slots and introduces constraints on how

these slots can be constructed:

$$Track = \langle Sl_1, \ldots, Sl_i, \ldots, Sl_n \rangle, Sl_i \in Term_{D\ EP}^{\mathcal{D}} \tag{7.12}$$

where $Term_{D\ EP}^{\mathcal{D}} = \{\ldots, D_x^y, \ldots\} \cup \{\ldots, EP_i, \ldots\} \cup \{\ldots, |EP_j|, \ldots\}$ is the union of the sets of all domains and all EPs within the domain tree $\mathcal{D}$ and all EP intervals that can be constructed based on these EPs. Sl_i is a constraint for the ith slot. If it is specified as an EP (or an EP interval), the corresponding slot in the CEP should be constructed based on an interval of this EP (or based on this EP interval itself). If it is a domain, this slot should be constructed based on an interval of an EP within this domain□

The Initial track (ITr) is a track that is provided by the user as the CEP structure requirements. So a slot's constraint in the ITr $Sl_i \in Term_{D\ EP}^{\mathcal{D}}$ restricts the set of EP intervals which can be used in order to construct the corresponding slot in the CEP. Therefore, the ITr defines the number of transfers that the student should make during the education according to the CEP and their high-level specification: from which domain to which domain the transfer should be done. From the structural perspective, the planning process can be represented as follows. Initially, based on the provided ITr, corresponding EP intervals are chosen for all slots. The resulting sequence of EP intervals is called a Basic Track (BTr). During the further planning, this BTr is refined into a concrete educational route for a student, where optional modules are chosen and some modules can be removed. So from the structural point of view, the planning is seen as a refinement of the ITr. At each step of the planning, the current track that was derived from the ITr and that represents the current solution is designated as $Track$.

In addition to the structural requirements specified as ITr, the set of EPs that can be used for solving a concrete CEP development problem is also limited. Within the domain tree $\mathcal{D}$, only EPs that can be used for the CEP construction are specified.

The CEP *property-requirements* determine specific characteristics of the future CEP. The following property-requirements can be specified by the user:

$$Req^{Prop} = \langle Award^{Req}, t_{Beg}, t_{End}, \delta^t \rangle \tag{7.13}$$

where $Award^{Req} = \langle Prov_{aw}^{Req}, Area, Lev_{Name}^{Req}, Lev_{Scale}^{Req} \rangle$ are requirements to the award that the student should receive as a result of the education at the CEP. $Prov_{aw}^{Req} = D_x^y$ is a domain where education provider $Prov_{aw}$ issuing this award should be situated: $Uni_{aw} \subseteq Uni_{Req}$. Since domains from different levels of the domain tree can be used, this requirement can be represented flexibly on different levels of abstraction, for example, specific countries, regions or universities can be used. Lev_{Name}^{Req} and Lev_{Scale}^{Req} determine the educational level that this award should cor-

respond: the level name Lev^{Req}_{Name} and the qualification framework identifier Lev^{Req}_{Scale}. If this requirement is specified as a level in the EHEA QF, all awards that correspond to NQF levels being equivalent to this level can be used. If a level in a NQF is specified, only awards at this NQF level can be used. *Area* is the field of study that the required award should refer to. Other property-requirements determine temporal constraints for this CEP. t_{Beg} and t_{End} are the earliest begin time and the maximum end time for the CEP. δ^t determines the maximum distance in months between two semesters taken sequentially in CEP. Hence, it specifies the maximum 'idle time' between adjacent slots. In addition to the property-requirements, for the CEP generation problem statement, properties of the student that will be enrolled in the CEP being developed are provided: $Student = \langle Stud_{Name}, Stud_{Country}, Lang_{Stud}, Hist_{Stud} \rangle$, where $Stud_{Name}$ is the identifier of the student, $Stud_{Country}$ is the student's country of origin, $Lang_{Stud} = \{\langle Lang_{Name\ i}, Lang_{Val\ i}, Lang_{Scale\ i} \rangle\}$ is the set of languages that the student knows at the corresponding level: $Lang_{Name\ i}$ is the language identifier, $Lang_{Val\ i}$ is the level of the knowledge according to the scale $Lang_{Scale\ i}$. $Hist_{Stud}$ is the history of student's education including HE EPs and pre-HE certificates. Details about the data model used for the student's properties specification are given within case studies in Chapter 9.

The CEP *process requirements* are specified as an initial task network $Req^{Proc} = TN$, that will be decomposed during the planning in order to produce a fully specified CEP process model. Planning tasks in the CEP generation framework represent how certain parts of the track are implemented, for example, which EP interval is used for its construction and what is the role of this part of the track in the educational (mobility) scenario. Therefore, these tasks link slots of the track, LObjs used for their construction and mobility scenarios that are utilised in the CEP being constructed. In the next sections, details about planning tasks used during the CEP generation and about the specification of the initial CEP process requirements, viz., the initial task network, are given. Additionally, these sections describe how CEPs are constructed using these tasks and decomposition methods designed for them.

7.3.1.2 BTr development phase

In the first phase of the CEP development, a BTr is constructed based on the process, structure and property requirements specified for the CEP. Planning tasks that are used within the current task network during this CEP development phase are compound tasks that relate a part of the track, an EP interval and a student studying according to the CEP. Such task assigns to this part of the track a specific role within the mobility programme being developed. Correspondingly, planning tasks used in this CEP development phase have the following parameters: $Student, Track(N^{Sl}_e, N^{Sl}_i), EP_{par}$, where the parameter *Student* represents the student that this CEP is being developed for, $Track(N^{Sl}_e, N^{Sl}_i)$ represents a part of the track *Track* from the $N^{Sl\ th}_e$

slot to the N_i^{Sl} th slot, EP_{par} is an optional parameter representing a set of EP intervals that can be used to construct the part of the CEP represented by the parameter $Track(N_e^{Sl}, N_i^{Sl})$. The EP_{par} parameter can be specified as an EP interval, then within this part of the track this EP interval should be studied, or as $Award^{Req}$, then an interval of EP that satisfies these award requirements should be studied within this part of the track.

Higher-level planning tasks used in the BTr development phase are tasks $Degree$, $Start_Degree$, $Start_Degree_Probation$, $Finish_Degree$, $Finish_Degree_fin$ and $Proceed_Degree$. The task $Degree(Student, Track, Award^{Req})$ is the highest level task that refers to the whole track $Track$. Hence, this task represents the whole CEP being developed. Correspondingly, the current task network that contains this task cannot contain other tasks. This task specifies that at the end of the education the student $Student$ will receive an award that satisfies requirements $Award^{Req}$. The structure of the CEP should conform with the structure requirements specified by the track $Track$. The task $Start_Degree(Student, Track(1, N_e^{Sl})[, |EP|_{[1,m]}])$ represents the initial part of the CEP, being developed for the student $Student$, that is covered by the part of $Track$ from its first slot to the N_e^{Sl} th slot. The start point of this task is an admission of the student to an EP. At the end point of this task, the student transfers from this EP to a new EP permanently, that is, he (or she) will not return to this EP back. The optional parameter $|EP|_{[1,m]}$ indicates the EP interval that the student is admitted to and that he (or she) transfers from. It is possible that during the execution of this task the student changes EPs several times and studies in other universities. The task $Finish_Degree(Student, Track(N_s^{Sl}, N_e^{Sl}), Award^{Req})$ represents the final part of the CEP, being developed for the student $Student$, that is covered by the part of $Track$ from the N_s^{Sl} th slot to the N_e^{Sl} th slot. The start point of this task is the student's transfer to an EP whose award satisfies requirements $Award^{Req}$. At the end of its execution, the student is graduated from this EP and receives the corresponding award. During the execution of this task, the student can transfer from and to this EP several times. Another high-level task is $Proceed_Degree(Student, Track(N_s^{Sl}, N_e^{Sl})[, Award^{Req}])$ that represents the intermediate part of CEP, being developed for the student $Student$, that is covered by the part of $Track$ from the N_s^{Sl} th slot to the N_e^{Sl} th slot. The start point of this task is the student's transfer to an EP[9] that he (or she) has not been admitted to and not the EP the student intends to graduate from. At the end of its execution, the student transfers from this EP. $Award^{Req}$ are requirements for the award that the student should receive at the end of the education. EPs that are used in all but the last slot of the CEP should not satisfy all these requirements. Indeed, these award requirements are used to limit the set of EPs that can be used in these slots. All these EPs should have awards in the same area as specified in $Award^{Req}$ and they should have levels equal or equivalent to the educational level specified in $Award^{Req}$ (this level can be specified in EHEA

[9]The student can transfer to this EP for the first time, or he (or she) can return to this EP.

QF or in some NQF). The tasks *Start_Degree_Probation* and *Finish_Degree_fin* are used to specify the temporal transfer scenario, when the EP that the student is admitted to and that he (or she) will graduate from is the same. They are used to specify the initial and the final parts of the CEP, while intermediate tasks, for example, the *Proceed_Degree* task is used to specify the educational process between them. Other tasks used in the BTr development phase are described in Appendix B (they are all compound tasks specified there).

The role of BTr development tasks during the planning is to define the mobility scenario that will be used in this CEP, map this scenario to specific parts of the track and select EP intervals that will be used in these slots. As tasks used in this phase define roles of corresponding parts of the track, these roles are defined only in relation to other tasks used in the same task network. So the task networks used in this CEP development phase are fully ordered, meaning that the initial task network is fully ordered and all decomposition methods contain only ordered task networks. The order of tasks in these task networks should correspond to the order of track intervals that they represent. Additionally, it is required that tasks utilised in task networks during the planning cover the whole track and do not intersect. So adjoining tasks in a task network should refer to adjoining parts of the track. If a task network is $\langle \dots, t_1(\dots Track(N^{Sl}_{s1}, N^{Sl}_{e1}) \dots),$ $t_2(\dots Track(N^{Sl}_{s2}, N^{Sl}_{e2}) \dots) \dots \rangle$, then track parts $Track(N^{Sl}_{s1}, N^{Sl}_{e1}$ and $Track(N^{Sl}_{s2}, N^{Sl}_{e2})$ should be adjoining and $N^{Sl}_{s2} = N^{Sl}_{e1} + 1$. These requirements are applied to task networks used as initial CEP process requirements Req^{Proc} and to tasks networks being produced during the planning. For example, the task network in Formula 7.14 represents a mobility scenario with one permanent transfer: the transfer after which the student will not return to the previous EP again.

$$\langle Start_Degree(S, Track(1,3), |EP|_{[n,m]}), Finish_Degree(S, Track(4,4), Award^{Req}) \rangle \qquad (7.14)$$

BTr development tasks used in a current task network can refer to parts of the track containing different number of slots. For example, in Formula 7.14, the task *Start_Degree* covers the part of the track from the first to the third slots. Such BTr development tasks are decomposed during the planning. As a result, new transfers are introduced and tasks covering less number of slots are produced. In the BTr development phase, the lowest level tasks are tasks that cover only one slot of the track and, in some cases, contain EP intervals that will be studied in these slots. Based on these tasks, the BTr is determined: it is the sequence of EP intervals used in tasks in the current task network. Such tasks will be referred as the BTr specification tasks.

Decomposition methods used in the BTr development phase represent a specific basic student mobility scenario being applied in a specific part of the CEP. The student mobility scenario can be a a permanent transfer, when the student does not return to the previous EP, or a temporal transfer, when the student returns back to the previous EP, viz., the probation period scenario. Different

decomposition methods are specified for the following parts of the CEP: an initial part of the CEP, when the EP that the student has been admitted to is concerned, an intermediate part or a final part, when an EP that the student will graduate from is concerned. Different methods for the same student mobility scenarios being used in different parts of the CEP are required because the contexts of these scenarios are different. For example, whether a student will eventually graduate from the EP that he (or she) transfers to or whether a student will return and graduate from the EP that he (or she) transfers from. Correspondingly, actions that will be used to implement these scenarios in lower level task networks are different.

The highest-level task *Degree* can be decomposed using three methods that implement three different mobility scenarios. One permanent transfer scenario (see Formula 7.15) introduces the task network consisting of *Start_Degree* and *Finish_Degree* tasks (for example, like in Formula 7.14, if the track consists of 4 slots). Two permanent transfers scenario (see Formula 7.3.1.2) additionally introduces the task *Proceed_Degree* between them. This results in two permanent transfers that should be carried out during the education according to the CEP. This method represents the situation when a student between the EP that he (or she) has been admitted to and the EP that he (or she) will graduate from studies at one or more other EPs. The temporal transfer scenario (the probation period scenario) is represented in Formula 7.3.1.2. It is specified using three tasks: *Start_Degree_Probation* designating an initial period of study according to an EP before the probation, *Proceed_Degree* designating the probation period and $Finish_Degree_fin$ designating the period of the study at the EP after the students returns to it and before the graduation. Other decomposition methods used in the BTr development phase are described in Appendix B. These methods apply the temporal transfer scenario to tasks that have different roles in the higher-level mobility scenario, so they should be implemented using different lower level actions. Additionally, a method is introduced that realises the one permanent transfer scenario within parts of the CEP that are implemented by 'intermediate' EPs (i.e., not the starting or finishing EPs) (see Formula 7.18). It is applied to a *Proceed_Degree* task that covers more than two slots of the track. As a result of its execution, two *Proceed_Degree* tasks are introduced. The first task covers one slot and has the EP interval specified. The second task covers the rest of the slots and should be applied to an interval of EP which is different from the previous EP.

$$Degree(S, Tr, Award^{Req}) \rightarrow \tag{7.15}$$
$$\langle Start_Degree(S, Tr(1, F_1), |EP|_{[1,m]}), Finish_Degree(S, Tr(F_1 + 1, F_0), Award^{Req}) \rangle$$

A method that can be chosen to decompose a task during the planning is restricted by the size of the track interval that is covered by this task and corresponding domain constraints contained in the ITr. So it is checked if the part of the track covered by the task being decomposed can

be divided into smaller parts (in other word, if new BTr development tasks can be introduced) and it is checked if the tasks introduced can be carried out according to constraints in the ITr for the corresponding slots (for example, for the temporal transfer, it is checked that according to the domain constraints the student will be able to return the EP that he (or she) has transferred from). Examples of different decompositions for a *Degree* task covering a four slots track are presented in Figure 7.2. Cases A. and B. show a variant when initially one permanent transfer scenario is applied, then one of the introduced tasks is decomposed using the temporal transfer scenario method for the corresponding part of the CEP (suffixes *_str* and $_fin$ refine the role of the higher-level task by modifications introduced using the further applied scenario). Case C. shows how the task network produced using the high-level temporal transfer method can be decomposed using one permanent transfer for intermediate EPs method. Using the latter method, one permanent transfer is introduced between tasks referring to 'intermediate' EPs in this CEP. Additionally, there are complex mobility scenarios where primitive mobility scenarios applied to the initial and finishing parts of the CEP overlap, for example, as in the task network $\langle Start_Degree_str, Finish_Degree_str, Start_Degree_fin, Finish_Degree_fin \rangle$. In order to generate such task networks, a one-slot *Proceed_Degree* task generated during the decomposition of a *Start_Degree* task is implemented by the first task of the task network that can be produced during the decomposition of the *Finish_Degree* task, that is, the *Finish_Degree_str* task. Subsequent one-slot *Proceed_Degree* tasks within the task network produced by the *Start_Degree* task can be implemented by *Finish_Degree_ret* tasks[10]. For example, the *Proceed_Degree* task in the task network $\langle Start_Degree_str, Proceed_Degree, Start_Degree_fin \rangle$ can be decomposed into a *Finish_Degree_str* task. Then, the task *Finish_Degree* should be decomposed only to task networks that represent the final part of the higher-level temporal transfer mobility scenario, that is, to the task $Finish_Degree_fin$ or the task networks $\langle Proceed_Degree,\ Finish_Degree_fin \rangle$, $\langle [Proceed_Degree,]\ Finish_Degree_ret, Proceed_Degree,\ Finish_Degree_fin \rangle$. As it can be seen in Appendix B, some one-slot BTr development tasks do not contain EP intervals. In order to reduce the overall number of decomposition methods, for these one-slot tasks the decomposition methods that select EP intervals were joined with the BTr validation methods that should be applied to these tasks next (see Section 7.3.1.3).

Initial *CEP process requirements* Req^{Proc} can be specified as a high-level task *Degree* when the user does not want to impose any constraints on the CEP process. Then, all possible CEP processes can be generated based on the CEP structure and property-requirements. Alternatively, the sequence of BTr development tasks can be provided as input process requirements. Using it, the user can define which mobility scenarios can be used in specific parts of the CEP. For the

[10]The task *Finish_Degree_ret* represents an intermediate part of the education at the EP that the student will graduate from. At the start point of this task, the student returns to this EP after studying at another EP. At the end, the student makes a temporal transfer from this EP. This task is described in more detail in Appendix B.

specification of the CEP process requirements, the described BTr development tasks are used in a restricted form: EP intervals are absent in their parameters. All information about possible EPs that can be used in these tasks is contained in the CEP structure, so the CEP process requirements are specified as complementary requirements relatively to the CEP structure requirements. The provided task network should define a correct mobility scenario and should comply with the CEP structure defined by the ITr. Thus, when the CEP process requirements are specified as a task network, at first, the planner tries to derive the task network with the same structure from the corresponding *Degree* task. If this succeeds, the planning is started from the produced task network.

$$Degree(S, Tr, Award^{Req}) \rightarrow \tag{7.16}$$
$$\langle Start_Degree(S, Tr(1, F_1), |EP|_{[1,m]}), Proceed_Degree(S, Tr(F_1+1, F_2), Award^{Req}),$$
$$Finish_Degree(S, Tr(F_2+1, F_0), Award^{Req})\rangle$$
$$Degree(S, Tr, Award^{Req}) \rightarrow \tag{7.17}$$
$$\langle Start_Degree_Probation(S, Tr(1, F_1), |EP|_{[1,m]}), Proceed_Degree(S, Tr(F_1+1, F_2), Award^{Req}),$$
$$Finish_Degree_fin(S, Tr(F_2+1, F_0), |EP|_{[n,k]})\rangle$$
$$Proceed_Degree(S, Tr(S_0, F_0), Award^{Req}) \rightarrow \tag{7.18}$$
$$\langle Proceed_Degree(S, Tr(S_0, S_0), |EP|_{[n,m]}), Proceed_Degree(S, Tr(S_0+1, F_0), Award^{Req})\rangle$$

Based on the CEP requirements, during the task network decomposition in the BTr development phase different complex mobility scenarios can be generated for the CEP being developed. Different scenarios can be produced depending on the methods applied, on the order of their application and on a chosen division of the track into intervals covered by different tasks. Additionally, during this decomposition, EP intervals are chosen based on the ITr constraints, award requirements and temporal constraints. Additionally, when EP intervals are selected, their educational content can be analysed in order to restrict the set of EPs that can be utilised for the construction of one CEP. Only similar EP intervals can be selected for the CEP construction based on the values of similarity measures defined in Section 7.2.2. As a result of the BTr development phase, a BTr is constructed as a sequence of EP intervals[11]. During the planning, the BTr development phase is not executed at once. As soon as a one-slot task that can be implemented first is produced, this task is decomposed further and lower level actions are produced initiating corresponding policy checks.

[11] With the exception for tasks that do not contain EP intervals. For these tasks, the EP interval will be selected during the next decomposition within the BTr validation phase

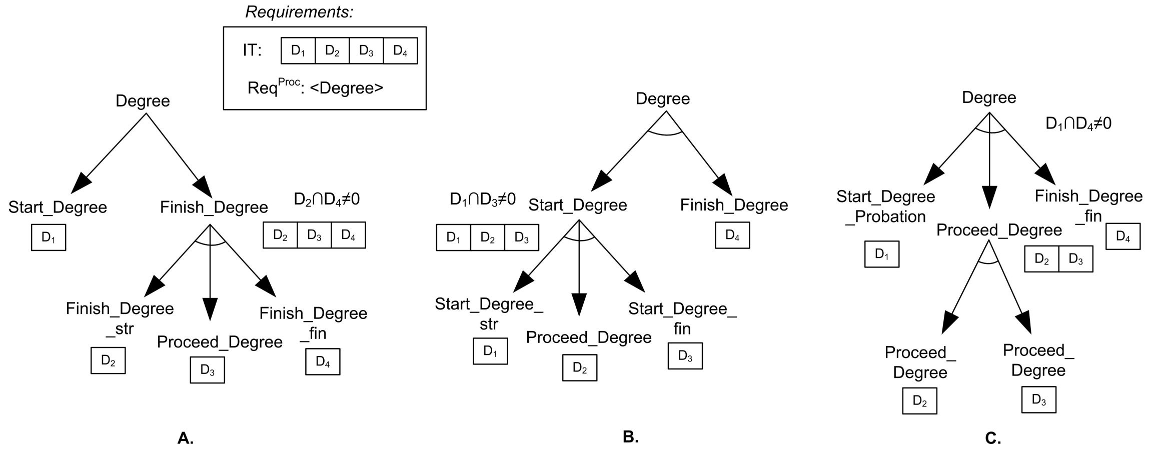

Figure 7.2: Examples of different decomposition methods application

7.3.1.3 BTr validation phase

In this phase, the BTr development tasks produced in the previous phase are decomposed one level lower, into primitive and compound actions. The produced task networks define at the EP intervals level a detailed CEP execution scenario that refines the BTr constructed. Actions used in this task network initiate corresponding policy evaluations, in order to validate the CEP constructed. Additionally, during their execution, the CEP is designed at lower level of detail: obligations are produced and compound actions are decomposed into lower-level tasks, initiating the further decompositions.

Generally, decomposition methods used in this phase decompose the tasks produced in the BTr development phase into lower level actions according to their definitions. Decomposition methods for tasks described in the previous section are represented in Formulae 7.19 - 7.23. The following actions were introduced in order to represent the execution of the BTr development tasks in this phase. These actions designate concrete situations that occur during the education of a student according to the CEP at the EP intervals level. Primitive actions !*admitP*, !*admitT* and !*graduate* designate the admission and graduation to/from CEP. Two actions for admission are used to distinguish situations when the student is admitted to an EP that he (or she) will graduate from (!*admitP*) or an EP that he (or she) will transfer from in the future at a permanent basis, that is, the student will not finish this EP (!*admitT*). Compound actions &*choose_modules* and &*study_interval* designate procedures for the optional modules selection and for studying an EP interval in a specific slot of the BTr. Additionally, several actions were introduced to designate incoming and outgoing transfers of a student to and from the EP interval. Incoming transfers are actions carried out when the student transfers to an EP interval. Outgoing transfers are, respectively, executed when the student transfers from an EP interval. Correspondingly, a pair of these action is used to model a student's transfer. Using the incoming and outgoing transfer

actions, policies for the home and destination universities can be evaluated separately. Outgoing and incoming transfer actions are divided into permanent and temporal transfer actions: they are designated using suffixes $transfer_IP$, $_OP$, $_IT$, $_OT$. Temporal outgoing and permanent incoming transfer actions $\&transfer_IP$ and $!transfer_OT$ are used when the student will graduate from the corresponding EP (for the outgoing transfer - the previous EP, for the incoming transfer - the next EP). Respectively, actions $\&transfer_IT$ and $!transfer_OP$ are used when the student will not graduate from the EP. These types of transfer actions are distinguished since during the execution of the former actions the university should check if it will be possible to award a degree for the student. Additionally, a specialised incoming transfer action $\&transfer_IRP$ is used to highlight the situation when the student returns to the EP that he (or she) has studied earlier and is going to graduate from it. For example, when the $Proceed_Degree$ task is decomposed, the first action is $\&transfer_IT$, which indicates that the student transfers to an EP that he (or she) will not graduate from, the last action is $!transfer_OP$, which indicates that the student transfers from this EP (but he (or she) can temporarily return to it in the future). Other methods used in this CEP development phase are specified in Appendix B.

$$Start_Degree(S, Tr(N^{Sl}, N^{Sl}), |EP|_{[n,m]}) \rightarrow \tag{7.19}$$
$$\langle !admitT(S, Tr, N^{Sl}, |EP|_{[n,m]}), \&choose_modules(S, Tr, N^{Sl}, |EP|_{[n,m]}),$$
$$\&study_interval(S, Tr, N^{Sl}, |EP|_{[n,m]}), !transfer_OP(S, Tr, N^{Sl}, |EP|_{[n,m]})\rangle$$
$$Finish_Degree(S, Tr(N^{Sl}, N^{Sl}), Award^{Req}) \rightarrow \tag{7.20}$$
$$\langle \&transfer_IP(S, Tr, N^{Sl}, |EP|_{[n,m]}), \&choose_modules(S, Tr, N^{Sl}, |EP|_{[n,m]}),$$
$$\&study_interval(S, Tr, N^{Sl}, |EP|_{[n,m]}), !graduate(S, Tr, N^{Sl}, |EP|_{[n,m]})\rangle$$
$$Proceed_Degree(S, Tr(N^{Sl}, N^{Sl}), [|EP|_{[n,m]} \ or \ Award^{Req}]) \rightarrow \tag{7.21}$$
$$\langle \&transfer_IT(S, Tr, N^{Sl}, |EP|_{[n,m]}), \&choose_modules(S, Tr, N^{Sl}, |EP|_{[n,m]}),$$
$$\&study_interval(S, Tr, N^{Sl}, |EP|_{[n,m]}), !transfer_OP(S, Tr, N^{Sl}, |EP|_{[n,m]})\rangle$$
$$Start_Degree_Probation(S, Tr(N^{Sl}, N^{Sl}), |EP|_{[n,m]}) \rightarrow \tag{7.22}$$
$$\langle !admitP(S, Tr, N^{Sl}, |EP|_{[n,m]}), \&choose_modules(S, Tr, N^{Sl}, |EP|_{[n,m]}),$$
$$\&study_interval(S, Tr, N^{Sl}, |EP|_{[n,m]}), !transfer_OT(S, Tr, N^{Sl}, |EP|_{[n,m]})\rangle$$
$$Finish_Degree_fin(S, Tr(N^{Sl}, N^{Sl}), |EP|_{[n,m]}) \rightarrow \tag{7.23}$$
$$\langle \&transfer_IRP(S, Tr, N^{Sl}, |EP|_{[n,m]}), \&choose_modules(S, Tr, N^{Sl}, |EP|_{[n,m]}),$$
$$\&study_interval(S, Tr, N^{Sl}, |EP|_{[n,m]}), !graduate(S, Tr, N^{Sl}, |EP|_{[n,m]})\rangle$$

Primitive and compound actions introduced at this level are provided with policy parameters tuples that have the uniform structure. For example, for the action $\&study_interval(Student,$

$Track_i$, N^{Sl}, $|EP|_{[n,m]}$), where N^{Sl} is the number of the slot in the track $Track_i$ that this action corresponds to, the following policy parameters tuple is defined: $\langle\{\langle Student$, $Subject\rangle$, $\langle EP$, $Resource\rangle\}$, $\{\langle N^{Sl}$, $Slot\rangle$, $\langle Track_i$, $Track\rangle$, $\langle n, Interval_start\rangle$, $\langle m$, $Interval_end\rangle\}\rangle$, where student object is used as a subject, EP object is used as a resource and numbers of the start and finish semesters for the EP interval are used as action attributes, along with the number of corresponding slot. For all actions, excepting the &*study_interval* action, start and end time points are the same. For the &*study_interval* action, duration is equal to the duration of the corresponding EP interval (i.e., the sum of durations of corresponding semesters). When actions introduced at this phase are executed, their effects are saved into the planner's world state. These effects represent the history of the student's education according to the CEP and are saved in relation with the corresponding slots of the track. For example, when the action &*study_interval* is executed, a history literal with the predicate symbol '*history*' is added into the planner's world state: $history(Student, Track_i, N^{Sl}, EP, n, m)$, where $Track_i$ is the track being developed during the plannings and N^{Sl} is the number of the slot in this track. In addition to these literals, other important milestones of the student mobility scenario execution are saved in the planner's world state, for example, the fact that a probation period was started. These effects are used for operators and methods precondition evaluation (e.g., in order to determine if an EP was already studied by the student, so it is prohibited to select it in some subsequent tasks). Moreover, these effects are used during the policy evaluation in order to derive a policy decision based on the history of the student's education. Effect for actions that are generated during this phase are specified in Appendix B.

7.3.1.4 Low-level routines

In the next phase of the CEP development, based on the primitive and compound actions produced during the decomposition of the BTr development tasks, routines for the design of lower level CEP processes are initiated. These processes operate at the modules level and can be classified into compulsory and optional processes.

Compulsory processes, which should be carried out in any CEP, are generated using routines initiated using decomposition methods for compound actions &*choose_modules* and &*study_interval*. Before an EP interval can be studied, all groups of optional modules should be closed, meaning that optional modules should be chosen such that corresponding policy requirements, governing their selection, are satisfied. As within an EP interval there can be several groups of optional modules and within one group there can be different number of modules, methods for the &*choose_modules* task are carried out using recursion. At the higher level, all groups within the EP interval are considered sequentially in a recursive manner using the method for the $Choose_modules_find_groups$ compound task. At the lower level, for each group different vari-

ants for the modules selection that results in the group closure are tried using the method for the $Choose_modules_group$ compound task. Therefore, all possible combinations of the optional modules selection are tried (see Appendix B). When a module is selected, the action !*choose_module* containing this module as a parameter is carried out. When a group is closed, action !*close_group* is carried out. The &*study_interval* compound action is also implemented using two recursive methods. The higher-level method for the $Find_sems$ compound task sequentially processes semesters within the considered EP interval. The lower-level method for the $Find_modules$ compound task processes core and selected optional modules within one semester (see Appendix B). Actions $!study_mod(Student, |EP|_{[n,m]}, Sem_i, Mod_j, N^{Sl}, Track_k)$ are used to denote the fact that module Mod_j within the semester Sem_i of $|EP|_{[n,m]}$ was studied by the student $Student$ at $N^{Sl\ th}$ slot of the track $Track_k$. These actions are carried out sequentially during the planning, while during the education they are studied in parallel. Obviously, this process is perceived more naturally using parallel actions in a resulting plan. So we introduced ‘pseudo-parallel’ actions. These actions are modelled during the planning sequentially and are connected using their effects and preconditions, being evaluated against the current planner’s world state. However, in the resulting plan, these tasks are represented as parallel actions. This is modelled using auxiliary actions !*concur_start*, !*concur_end* and !*change_line*, which are inserted into the plan using decomposition methods in order to mark concurrent parts of the plan. !*concur_start* and !*concur_end* designate start and end points of a segment with parallel plan sections. !*change_line* is used to designate the end of one parallel section and beginning of the next section. When the resulting plan is generated, these auxiliary action are analysed and are used to convert this sequential plan into the plan with parallel sections (see example in the second case study in Chapter 9).

Execution of optional processes and task networks that represent them is initiated using obligations. Obligations are specified by policy authors as part of the policy specification. In order to generate a correct CEP process, obligations should be triggered only in certain situation, when these obligations can be executed according to the educational process. In order to restrict a set of obligations that can be generated during the evaluation of a policy request with certain action, corresponding obligations validation rules were designed as part of the planning domain specification (see its subset in Figure 7.3). Since all these rules are valid for the whole domain, asterisk symbol is used in their policy lists.

The first and second obligation validation rules specify that in order to admit a student to a university, he (or she) might need to pass exams. So the admission action can be augmented with the !*sit_exam* obligation. Another educational routine for which different variations exist is a modules prerequisites evaluation routine. Generally, for studying a LObj by a student, either all its prerequisites should be satisfied or only certain part of the prerequisites should be satisfied. Another variation is a level in the EP structure at which the number of unfulfilled prerequisites

1. $((!admitP\ (before\ (!sit_exam)) => *)$
2. $(!admitT\ (before\ (!sit_exam)) => *)$
3. $(!study_mod\ (before\ (\&check_prereq_mod)) => *)$
4. $(\&check_prereq_mod\ (during\ (Satisfy_prereq_mod_all)) => *)$
5. $(\&check_prereq_mod\ (during\ (Satisfy_prereq_mod)) => *)$
6. $(\&study_sem\ (before\ (\&evaluate_precond_sem))(after\ (!ic_prereq_sem)) => *)$
7. $(\&study_sem\ (before\ (\&evaluate_precond_sem_all)) => *)$
8. $(\&recognise\ (during\ (Recognise_1_1)) => *)$
9. $(\&transfer_IP\ (during\ (ordered\ \&recognise\ !discard_difference)) => *)$
10. $(\&transfer_IP\ (during\ (ordered\ \&recognise\ \&move_to_sem)) => *)$
11. $(\&transfer_IP\ (during\ (ordered\ \&recognise\ \&evaluate_difference)) => *)$
12. $(\&transfer_IP\ (during\ (ordered\ \&recognise\ \&evaluate_difference_intermed\ \&move_to_sem)) => *)$
13. $(\&transfer_IP\ (during\ (ordered\ \&evaluate_difference)) => *))$
14. $(\&transfer_IP\ (during\ (ordered\ !discard_difference)) => *)$

Figure 7.3: Obligation validation rules

is aggregated and evaluated. For example, there can be constraints that a certain percent of prerequisites should be satisfied for each module, or this percent is calculated for modules within the current semester or for all modules within the current EP interval. If prerequisites are evaluated separately for each module, a before-obligation action $\&check_prereq_mod$ should be generated for the action $!study_mod$ (see the third rule). For this action, a during obligation should be used to specify which specific procedure should be used to evaluate prerequisites[12]. These procedures are represented by the task $Satisfy_prereq_mod$ or task $Satisfy_prereq_mod_all$ (see rules four - five). The former task specifies that some module's prerequisite can be not satisfied, while the latter task requires that all prerequisites should be satisfied. During the execution of the task $Satisfy_prereq_mod$, action $!satisfy_prereq_mod$ is executed for each prerequisite of the module. Policies for this action specify conditions for fulfilment of this prerequisite. If this action cannot be executed, the planner continues planning. When all prerequisites for the module are evaluated, the action $!ic_check_prereq_mod$ is executed in order to estimate the overall number of satisfied and not satisfied prerequisites (specific threshold should be specified in conditions of corresponding policies) and derive a final decision if this module can be studied by the student. On the contrary, when the task $Satisfy_prereq_mod_all$ is used, further planning is blocked if some prerequisite is not satisfied. Another approach to the prerequisites evaluation requires that all prerequisites of modules in one semester should be evaluated within one procedure (i.e., without the division of prerequisites in groups corresponding to their modules). For this prerequisites evaluation, there are two possible compound actions that can be used as before-obligations: $\&evaluate_precond_sem$

[12]These obligations are separated to have the possibility to specify them in policies separately, for example, in different policies.

and &*evaluate_precond_sem_all* (see rules 6 - 7). Similarly with the previous pair of prerequisites evaluation tasks, the latter task requires that all prerequisites should be satisfied, while the former task requires just certain percent of satisfied prerequisites. During the decomposition of the former task, the additional after-obligation !*ic_prereq_sem* is used to estimate if the required number of prerequisites are satisfied. Examples of different prerequisites evaluation schemas are presented in Chapter 9.

Using obligations, policy authors can also define a routine for the execution of incoming students' transfers. During a student's transfer the following tasks can be carried out. Possible combination of these tasks are specified using the obligations validation rules 9 - 14 in Figure 7.3. First of all, the planner can try to recognise modules of the EP that the student transfers to. In this case, the compound action &*recognise* should be returned as a during-obligation for the action &$transfer_IP$. In turn, a during obligation for the compound action &*recognise* should specify which type of recognition can be used. In the current version of the CEP generation solution, only one-to-one modules recognition is supported (i.e., one module must be recognised by one another module) (see rule 8). During the execution of this task, modules are recognised using the action !$recognise_module_1_1(Mod_1, Mod_2, Student, Track_k, N^{Sl}, |EP|_{[n,m]})$, where Mod_2 is a recognised module, Mod_1 is a module studied by the student that is used to support the recognition. When a module is recognised, it is marked using a special flag. After the &*recognise* action, other tasks should be used that process the recognition results and introduce other possibilities to process modules, that were not recognised by the &*recognise* action. Using the compound action &*move_to_sem*, unrecognised modules that the student should have studied can be moved into the EP interval that the student transfers to. The second possibility is to recognise such modules using additional assessments. During the decomposition of the task &$evaluate_difference$, which is returned as a during-obligation, actions !*evaluate* are applied to such modules and it is checked if policies permit their assessment and recognition. At the end of the task networks produced during the &$evaluate_difference$ and &*move_to_sem* decomposition, action !$discard_difference$ is used to estimate if modules that were not recognised and were not moved into the EP interval can be neglected and the transfer can still be carried out. The action !$discard_difference$ can also be returned as an independent during-obligation (see rule 9), when if the during obligations set for the action &$transfer_IP$ does not contain obligations &$evaluate_difference$ and &*move_to_sem*. An example of the &*recognise* action execution is presented in Chapter 9.

Primitive and compound actions used in this phase have different policy parameters tuples structures, but the general approach is that the student is used as a subject and the main LObj that the current action is applied to (it can be a module, a semester or an EP) is used as a resource. If other LObjs are used as parameters of this task, they are represented as additional designated objects with corresponding roles. For example, in the action !$recognise_module_1_1(Mod_1$, Mod_2,

$Student$, $Track_k$, N^{Sl}, $|EP|_{[n,m]}$) module Mod_1, that is, the module used to support the recognition, is used as a designated object with the role '$ModToSupport$' and module Mod_2, that is, the recognised module, is used as a designated object with the role '*Resource*'.

The resulting plan represents a detailed CEP process that should be executed during the education of the student according to the resulting CEP. As this plan is hierarchical and each semester and EP interval is designated by compound actions &*study_sem* and &*study_interval*, the CEP structure can be easily extracted. Modules of this CEP are contained as parameters in actions !*study_mod*. They can be easily grouped into semesters, as actions with modules that belong to the same semester are carried out within the same &*study_sem* compound action. Optional modules used in this CEP can be distinguished by the !*choose_module* actions, performed during their selection.

7.3.1.5 Variations of overall CEP construction procedure

Two variants of the CEP generation procedure were designed. In the first variant, after the user specifies the CEP requirements, the planner carries out all three phases of the CEP development and produces a fully specified CEP for the user. The drawback of this variant is the fact that the user should wait until the planner develops a fully specified CEP that can be returned for his (or her) evaluation. Moreover, the user can be not satisfied with the produced CEP, for example, because he (or she) have not specified all the CEP requirements that he (or she) intended. In order to overcome this drawback, the second variant of the CEP construction procedure was introduced. In this variant, the user is provided with preliminary results of the planning: these are BTrs constructed in the first phase of CEP generation and validated at its second phase (see Sections 7.3.1.2 - 7.3.1.3). As was shown, BTrs are used as a basis for the further generation of detailed CEPs, so they provide general information about these CEPs. The BTrs can be constructed faster for the user, based on them the user can intervene into the planning process. He (or she) can modify input requirements and run the BTr development and validation phases again, if unsatisfactory or empty results were received. Alternatively, the user can select one or several BTrs that he (or she) prefers that will be passed to the next phase of the CEP generation. The CEP generation solution designed can support both variants of CEP construction procedure: ordinary and with intermediate BTr results. They can be selected by the user based on concrete parameters of the problem and the planning environment used, specifically, based on the scale of the problem. The second variant, when initially only BTrs are designed, is advantageous for the large scale problems as the user can intervene into the CEP construction at the earlier stages and can guide the planning process. As a result, CEPs that satisfy the user more can be produced. Moreover, in addition to the overall optimisation of the CEP construction process, a performance improvement technique was designed specifically for BTr development and validation phases (see Section 7.3.2).

7.3.2 Descending policy evaluation technique

During the BTr development and validation phases, a large search space should be explored if high level CEP requirements have been specified and a large set of EPs that can be used for the CEP construction has been provided to the system. The crucial difficulty is the search for EP intervals that can be used for the construction of a CEP in a way that all policy constraints specified for this CEP are satisfied. Policies specified at different levels of the hierarchical multi-domain structure impose limitations on the consistent use of these EPs. The postponed policy enforcement mechanism, which was developed in Chapter 6, provides the means to improve the performance of policy-based planning by earlier evaluation of policies during the planning and, correspondingly, to generate results of the planning in less time. In this section, the postponed policy enforcement mechanism is applied to the BTr development problem, in concrete, to the EP interval search process. As a result, a problem-specific technique for the planning performance improvement is developed. This technique is the descending policy evaluation.

7.3.2.1 Utilisation of postponed policy enforcement for CEP generation problem

The CEP generation planning domain was extended in order to utilise the advantages of the postponed policy enforcement. For this purpose, first of all, high-level effects, which can represent the known part of future effects, and partial policy vectors, representing known information about policy request parameters, should be defined. The core process of the BTr development is the selection of EP intervals for slots in a track, so these EPs and their positions in the track are used as high-level effects of compound tasks in the BTr development phase. EP intervals can be represented as Dummy objects that have properties and relations representing the known part of their specification. In the initial stages of the planning, this information can be derived based on the corresponding constraints specified in the ITr and requirements for an award that should be received at the end of the education.

For example, if a higher-level *Degree* task is provided as an initial task network, it is known that the last slot of the track will be constructed based on EP that has an award with a level that is equal or equivalent to the level requirements $\langle Lev^{Req}_{Name}, Lev^{Req}_{Scale}\rangle$. It is known that this award should have problem area equal to the *Area* parameter provided as a requirement for the resulting award. Moreover, any EP that can be used in this slot will be within a domain D_{lower}, that is, a lower level domain between the domain used in the corresponding slot of the ITr and the domain used in the award's requirements. In this technique, in addition to the domain tree hierarchy of properties, we have defined a set of property hierarchies for educational levels: levels in the EHEA QF are used as roots in these hierarchies and levels in NQFs equivalent to them are used as leafs. Hence, the Dummy EP for the last slot in the BTr can be defined as $EP_{Dm_1} = \langle \{\}, \{, \langle D_{lower}, \{Area\}, \langle Lev^{Req}_{Name}, Lev^{Req}_{Scale}\rangle\rangle\}, D_{lower}\rangle$. The fact that the

set of semesters is empty and there is only one area in the area set (while a real EP might have more than one area) conforms with the definition of a Dummy object. During the processing of this object, it will be considered that any other properties and relations can be added to it. So we have updated the *Degree* task processing in the following way. If this task is specified as an initial task network, before the actual planning starts, a high-level effect is assigned to this task. This high-level effect designates the history literal referring to the last slot of the track that will be added into the planner's world state when the corresponding &*study_interval* will be executed: $history(Student, Track_i, n, EP_{Dm_1}, NIL, NIL)$, where n is the number of the last slot in $Track_i$. Obviously, in addition to the high-level effect, partial policy vectors can be assigned to this task. These partial policy vectors represent actions !*graduate* and &*study_interval* because for the *Degree* task it is known that these actions will be carried out during each possible decomposition. For example, the partial policy vector for the !*graduate* action is the following: $\langle\{\langle Student, Subject\rangle, \langle EP_{Dm_1}, Resource\rangle\}, !graduate, \{\langle n, Slot\rangle, \langle Track_i, Track\rangle, \langle NIL, Interval_start\rangle, \langle NIL, Interval_end\rangle\}, TInterval^P\rangle$, where $TInterval^P$ is a loose time interval equal to interval $\langle t_{Beg}, t_{End}\rangle$. It should be noted, other high-level effects and partial policy vectors for the task *Degree* cannot be specified, because according to the Dummy object definition each Dummy object is distinct, meaning that the same Dummy object-terms should be substituted by the same ordinary object-term. For the *Degree* task it is unknown where equal EPs will be used within a track, so only when further decomposition methods are applied, more high-level effects can be added.

BTr development methods described before were modified such that these methods add some specific new information for the partial policy evaluation when they are applied during the planning. They assign new high-level effects, add more partial policy vectors or refine existing high-level effects and partial policy vectors. For example, during the execution of method $Degree(\ldots Track(1, 4)\ldots) \rightarrow Start_Degree(\ldots Track(1,3)\ldots), Finish_Degree(\ldots Track(4,4)\ldots)$, illustrated in Figure 7.2, case B (first method), the following new structures should be assigned to new tasks being produced during the decomposition. Based on the task *Start_Degree*, it is known that EP intervals of the same EP will be used in slots one and three. Hence, a new Dummy EP can be specified for these slots: $EP_{Dm_2} = \langle\{\}, \{\langle, D'_{lower}, \{Area\}, \langle {Lev^{Req}_{Name}}', ehea_qf\rangle\rangle\}, D'_{lower}\rangle$, where D'_{lower} is the lower domain between domains in ITr for slots one and three. The educational level is specified as a level ${Lev^{Req}_{Name}}'$ in EHEA QF because awards of non-final EPs within a track should correspond to the EHEA QF level of the final award. Correspondingly, this Dummy EP is used in *history* high-level effects for slots one and three. Partial policy vectors with this Dummy EP can also be added to the task *Start_Degree*. These partial policy vectors refer to the first and third slots. For the first slot, partial policy vectors with actions &*study_interval*, !*admitT* and !$transfer_OP$ are added. For the third slot, partial policy vectors with actions &*study_interval*, !$transfer_OP$ and

$\&transfer_IT$ are added.

The illustrated principle is used for the specification of high-level effects and partial policy vectors for other methods also. When during the application of a method it is known that in some slots the same (or distinct) EPs are used, corresponding Dummy EPs can be specified and added into the method definition as new high-level effects and partial policy vectors that should be assigned during its execution to the produced tasks. Additionally, in order to satisfy the correctness requirement, each high-level effect and partial policy request of the task that this method is applied to should be propagated to lower level tasks or refined. Loose time intervals are equal to the interval for the *Degree* task ($\langle t_{Beg}, t_{End} \rangle$) with the difference that a current time is used as the start point of the interval if some actions were already executed[13].

Using this principle, when one-interval BTr specification task is produced during the decomposition, high-level effects and partial policy vectors of the task that has been decomposed are refined and a specific EP interval is determined. Then, corresponding high-level effects and partial policy vectors become equal to effects and policy vectors that will be assigned and evaluated by lower level actions. So, for these policy vectors, only permanent decisions are produced during the policy evaluation.

7.3.2.2 Descending policy evaluation algorithm

During the BTr development, it is required to explore a large set of EPs stored within the planning environment and select EP intervals that can be used in the BTr in a way that all policies are satisfied. The planning domain for the BTr development phase contains few methods, so the following property holds when the CEP requirements are specified loosely, meaning that domain constraints in the ITr and resulting degree are specified at higher levels of the domain tree. The average number of possible EPs that can be used for the instantiation of an EP-variable within a planning task ($n(EP)$) is much greater than the average number of methods applicable to this task ($n\mathcal{M}$) multiplied by the average number of possible instantiations of other variables used within the methods' preconditions ($n(Pre)$)[14]: $n(EP) \gg n(Pre) \cdot n(\mathcal{M})$. In such situations, the Fewest-Alternatives First (FAF) strategy can be adopted to improve the planning performance. This strategy is used for the selection of flaws for processing during the planning. It was shown that it can improve the planning performance in a broad range of HTN planning domains [156][15]. According to this strategy, the instantiation of EP-variables should be postponed during the planning (i.e., the least-commitment approach should be used for the EP-variables instantiation) and during the

[13] Additionally, after the execution of an action, a specialised procedure is used to update time intervals in partial policy vectors assigned to tasks in the current task network. When the *&study_interval* action is carried out, which represents the execution of an EP interval, start time points of possible time intervals in these partial policy vectors should be updated.

[14] For example, variables for the start and end semesters of the EP interval that will be used in the tasks produced.

[15] This strategy analyses the number of possible branches produced when certain flaw is resolved and selects the flaw that brings less branching. So in contrast to the static strategies, which select flaws based on their types, it selects preferable flaws dynamically based on a current situation.

application of methods EP-variables should be kept non-instantiated.

Using the CEP generation planning environment peculiarities, this principle can be extended. Based on the fact that policies are organised hierarchically according to the domains tree and are related with the Deny-overrides combining algorithm, it follows that a policy request containing an EP should be permitted by policies of all domains where this EP is included. Correspondingly, when it is required to select an EP for an EP-variable, initially a domain can be selected instead of selecting an EP value. This domain value represents an area where this EP will be situated. Namely, when the EP-variable will be instantiated by a specific EP, this EP should be selected from this domain. During the planning, the domain value for the EP-variable is refined: the domain value is substituted with lower-level domains nested in the original domain. So the domains are updated in a descending manner, limiting the EP search area. When a new domain for an EP is selected, partial policy requests referring to this EP and containing known information about it should be evaluated against the policies of this domain. When a partial policy request is evaluated into a permanent decision, this decision refers to the whole set of EPs situated within the current domain, because when the policy request is refined, the same decision should be produced for all these EPs. This provides the possibility to prune several EPs from the search space earlier during the planning when some request is denied, so improving the planning performance.

$DescPE$(${PolVec}^P$ - partial policy vector, EP_{Dm} - Dummy EP, D_{xi}^{yi} - domain or EP, S_{Req^P} - set of partial policy vectors)

1. Evaluate partial policy request for ${PolVec}^P$
2. **If** result is $Deny$ or $IndPerm$ **then Return** Failure **endif**
3. **If** result is $IndTemp$ **then** add a copy of ${PolVec}^P$ into S_Req^P **endif**
4. **If** result is $Permit$ or N/A and D_{xi}^{yi} is EP **then**:
   4.1. Re-evaluate all partial policy vectors in S_{Req^P} with known EP equal to D_{xi}^{yi}.
   4.2. **If** any $Deny$ has occurred **then Return** Failure **else Return** D_{xi}^{yi} **endif**
   endif
5. **Loop** for all children D_{xk}^{yk} of D_{xi}^{yi} in the domain tree:
   5.1. **If** D_{xk}^{yk} is EP **then** Check requirements to EP. **If** requirements are not satisfied **then Continue** with next D_{xk}^{yk} **endif endif**
   5.2. Update domain of EP_{Dm} with D_{xk}^{yk}, add EP_{Dm} into ${PolVec}^P$
   5.3. Assign $Res := DescPE$(${PolVec}^P$, EP_{Dm}, D_{xk}^{yk}, S_{Req^P}). **If** $Res \neq$ Failure **then** Return Res **endif**
   EndLoop
6. **Return** Failure

Figure 7.4: Basic descending policy evaluation algorithm

An algorithm for the descending policy evaluation is presented in Figure 7.4. The descending policy evaluation is carried out for partial policy vectors that were defined in Section 7.3.2.1. Each of these partial policy vectors ${PolVec}^P$ contains a Dummy EP (EP_{Dm}) that represents all known

information about the EP used as a designated object with the 'Resource' role and is attached to a one-slot task TA in a current task network. Before the algorithm is called, initial domain value $D^{x_i}_{y_i}$ for EP_{Dm} is determined. It is equal to a domain value in the corresponding slot of ITr. So as input, the procedure `DescPE` gets a partial policy vector $PolVec^P$, a Dummy EP EP_{Dm}, which is used in this vector, and a domain $D^{x_i}_{y_i}$, which is equal to the domain of EP_{Dm}. Additionally, a set S_{Req^P} is provided as an input for the procedure `DescPE`. When this procedure is called for the first time for a policy request and an EP, this set is empty. During the recursive calls, it is populated with postponed policy request vectors. The procedure is carried out recursively searching for an EP in a domain tree in a descendant manner: in each cycle an input partial policy vector is evaluated, results of the evaluation are processed and a child value of the domain $D^{x_i}_{y_i}$ is chosen as a new domain for the Dummy EP EP_{Dm}. The policy vector $PolVec^P$ is evaluated only against policies in a current domain $D^{x_i}_{y_i}$. When this partial policy request is permitted, this means that these policies should not be evaluated further for any EPs that will be found within this domain. If it is denied, a backtrack should be done as this means that this request will be denied for all these EPs. If $IndTemp$ decision is returned for this request, this means that a permanent decision cannot be determined for this request and the request should be saved into the S_{Req^P} set and re-evaluated when a concrete EP will be selected. So if the partial policy request is permitted and the current value of $D^{x_i}_{y_i}$ is EP (and all policy vectors in S_{Req^P} were permitted) this EP is returned as a result. It should be also noted that the returned EP should satisfy the user requirements, so in step 5.1., when a new domain value $D^{x_i}_{y_i}$ is selected and it is an EP, it is checked if this EP satisfies the user requirements. In Figure 7.4, the descending policy evaluation procedure was illustrated as an example for one partial policy vector and for a one-slot task, meaning that the constraint on its domain specified in the ITr can be immediately enforced. The complete version of the descending policy evaluation procedure will be considered in the next sections.

7.3.2.3 Domain refinement

The performance of planning depends on the possibility to make critical decisions earlier and postpone other decisions until a good opportunity occurs [154]. According to this principle, the descending policy evaluation procedure was split into several stages and these stages are carried out in different steps of the planning. Each stage corresponds to the evaluation of policies in one domain. So finer-grain decisions can be taken during the planning. The descending policy evaluation procedure can be interrupted at a certain stage and the planner can switch to the decomposition of tasks within the current task network. One stage in the descending policy evaluation is realised as a domain refinement operation. In order to implement these operations, the original planning environment is modified. All tasks are specified as localised tasks $TA^D := \langle task(TA^D), domain(TA^D)\rangle$, where $task(TA^D)$ is an original task atom, $domain(TA^D) \in Term^{\mathcal{D}}_{D\ EP}$ is a domain value for the

task $task(TA^D)$ that indicates that it should be executed within the domain $domain(TA^D)$ (that is, the descending policy evaluation algorithm has stopped at this domain for the task TA^D). So EPs used in all tasks generated during the decomposition of the task TA^D should be within the domain $domain(TA^D)$.

When the planning starts, no restrictions are specified on the domains of tasks within the initial task network, that is, a root domain is used as domain value. Methods in the descending policy evaluation are divided into two disjoint sets: decomposition methods and domain refinement methods. Decomposition methods are produced based on the BTr development methods, used to the solve the BTr development tasks (see Section 7.3.1.2). The following updates were introduced for these methods: during the decomposition, they do not instantiate EP variables and do not modify the tasks' domains. The decomposition of tasks in a current task network using these methods is represented in an abstract way using the function $R_E(TA_0^D, N_0)$. This function represents which updates to a current state of the planner can be introduced when certain task in a current task network is decomposed by some decomposition method. So this function is applied to state N_0, a state of the planner formed by the current planner's world state $PlanState(N_0)$ and the current task network $tasks(N_0)$ ($N_0 = \langle PlanState(N_0), tasks(N_0)\rangle$), and to task TA_0^D, a task in the current task network $tasks(N_0)$. This function returns a set of alternatives states $\{N_1, \ldots N_i \ldots, N_k\}$ produced as a result of the TA_0^D decomposition: the task TA_0^D is substituted in these states by alternative task networks TN_i[16]. Constraints on these states, imposed by the updates introduced for decomposition methods, are specified in Formula 7.24. In each produced state N_i, within the task networks TN_i that substituted the task TA_0^D domains for all tasks are equal to the original domain of the task TA_0^D. Additionally, according to the postponed policy enforcement, during the application of new methods, new partial policy vectors and high-level effects are generated and existing vectors and high-level effects are refined, as was described in Section 7.3.2.1.

$$\begin{gathered} R_E(TA_0^D, N_0) = \{N_1, \ldots N_i \ldots, N_k\} \\ \forall i \text{ (} N_i \text{ is equal to } N_0 \text{ where } TA_0^D \text{ is substituted by } TN_i, \\ \forall TA_j^D \in TN_i \text{ (} domain(TA_j^D) = domain(TA_0^D) \text{))} \end{gathered} \tag{7.24}$$

Domain refinement methods are added into the planning domain as a new set of methods. Possible domain refinements carried out by these methods for a certain task in a current planner's task network are represented by the domain refinement function $R_D(TA_0^D, N_0)$. This function is applied to a current planner's state N_0 and task TA_0^D in a current task network $tasks(N_0)$. This function returns a set of alternative states produced when a domain refinement method is applied to the

[16]These alternative states are produced by different methods, applicable to the task TA_0^D, and different instantiations of variables in their preconditions.

task TA_0^D. In these states, the domain for the task TA_0^D in the current task network is updated by a new domain value, which is a descendant of its original domain. The rest part of the task TA_i^D and the rest of the task network is not modified by a domain refinement method (except in the situation when an EP for the task TA_i^D is selected instead of a new domain value, then the EP-variable within TA_i^D should be instantiated). The constraints on the produced set of states, imposed by a domain refinement method definition, are specified in Formula 7.25. Using these methods, the descending policy evaluation is implemented. When a domain refinement is executed, one step within the descending policy evaluation procedure is carried out. So during the domain refinement, partial policy vectors corresponding to the current task are evaluated in a new domain. Correspondingly, when an EP in found, the postponed partial policy vectors are re-evaluated based on new available information.

$$R_D(TA_0^D, N_0) = \{N_1, \ldots N_i \ldots, N_k\}$$
$$\forall i \text{ (} N_i \text{ is equal to } N_0 \text{ where } TA_0^D \text{ was substituted by } TA_i^D, \tag{7.25}$$
$$domain(TA_i^D) \subseteq domain(TA_0^D) \text{)}$$

Execution of operators is represented in an abstract way using the function $R_{Op}(TA_0^D, N_0)$. This function is applied to a current planner's state N_0 and a primitive action TA_0^D in a current task network $tasks(N_0)$. This function returns a set of alternative states produced when an applicable and legitimate operator is applied to the action TA_0^D. In the produced states, both the current planner's world state and task network are updated. The only exception to the standard operators execution procedure is the fact that during the descending policy evaluation an operator cannot be applied until the action task is not fully specified (including a concrete EP that is used instead of the Dummy EP). Compound actions and obligations are not supported in the current version of the descending policy evaluation algorithm, because in the BTr development phase of the CEP construction they are not used.

As was stated, the domain refinement is implemented using domain refinement methods. Three different types of domain refinement methods are designed. In the general case, for each compound planning task three domain refinement methods with different types should be specified[17]. Each method implements a specific phase of the descending policy evaluation process. Successive application of these methods to tasks in a current task network realises the descending policy evaluation algorithm. The domain refinement procedure introduced extends the basic algorithm for descending policy evaluation presented in Figure 7.4: it can be applied to any task, not only to a one-slot task.

[17]But as will be described later, these methods can be applied only when a specific constraint on this task is satisfied.

An abstract state transition diagram illustrating updates introduced to tasks in a current task network during the planning is presented in Figure 7.5. Abstract states of tasks are determined by three parameters of the task: number of slots that this task covers in the track ($N_{Sl}(TA^D)$), flag indicating if the domain constraint from the corresponding slot of the ITr was applied for this task (Enf^{ITr}) and, additionally, when Enf^{ITr} is false and $N_{Sl}(TA^D) = 1$, the third parameter is used to show if the condition $domain(TA^D) = ITr_1^{Constr}(TA^D)$ is satisfied. The latter parameter indicates if a current domain of the task $domain(TA^D)$ is equal to the domain constraint for this task in the ITr $ITr_1^{Constr}(TA^D)$[18]. It is required as there can be situations when the flag Enf^{ITr} is not set up but the ITr domain constraint for the task is already satisfied. For each state of a task determined by values of these parameters, a distinct type of domain refinement method is designed. This guarantees that for each task during the planning at most one domain refinement method can be applied. Figure 7.5 represents abstract states of tasks and how they are changed during the planning using the task decomposition and domain refinement. Updates of states carried out using task decompositions are designated as double arrows lines. During the decomposition, one task can be substituted by a set of new tasks, but all these tasks can be only in states that the decomposition arrow points at.

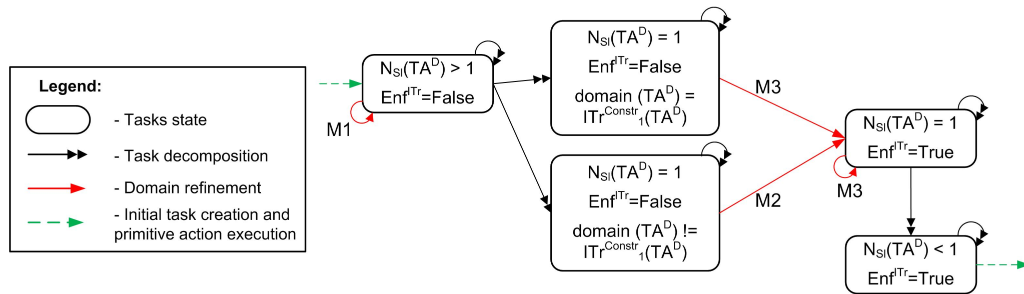

Figure 7.5: State transition diagram for task states during the planning

The domain refinement methods are shown in the figure as single arrowed solid lines. Domain refinement method $M1$ is applied to tasks covering several slots of the track. This method assigns a new domain value equal to the Least Common Ancestor (LCA) for domains that are used as constraints for slots covered by this task in the ITr. The number of slots $N_{Sl}(TA^D)$ that are covered by task TA^D is reduced during the task decomposition. So domain refinement methods $M2$ and $M3$ are applied to tasks corresponding to only one slot in the track. Method $M2$ enforces the domain constraint from ITr for this task: $domain(TA^D) := ITr_1^{Constr}(TA^D)$. So it is applied when current domain of the task is not equal to this domain constraint. Additionally, it sets the flag Enf^{ITr} to $True$. Method $M3$ substitutes the current task's domain with one of its children. It is applicable either when the flag Enf^{ITr} is set to $True$ or when the current domain is equal

[18] The function $ITr_1^{Constr}(TA^D)$ returns a domain constraint in the ITr for the slot that the task TA^D corresponds to. This function is determined only for one-slot tasks.

to the domain constraints from the ITr: $domain(TA^D) = ITr_1^{Constr}(TA^D)$. The second condition is required as during the application of method $M1$ the enforced LCA value could be equal to the ITr domain constraint for some slot covered by the TA^D task. In this case, during the later task decompositions, the task corresponding to this slot will be generated already with the domain equal to its ITr domain constraint. This situation is detected using method $M3$ and after its application the flag Enf^{ITr} is set to $True$, as required. All these domain refinement methods are specified in a way that they can be applied only when the corresponding conditions on the task state are satisfied and when the new domain value that they are going to assign is distinct from the current task's domain.

7.3.2.4 Algorithms for planning with domain refinement

The descending policy evaluation technique was developed for the BTr development phase of the CEP generation. During the CEP generation, only fully ordered task networks are used, so in this section we consider planning with ordered task networks: $tasks(N) = \langle TA_1^D, TA_2^D, \ldots, TA_n^D \rangle$, that is, ordered task networks are used in problem statements and ordered task networks are used in methods.

Domain refinement methods constitute a distinct type of methods that should be used alternatively with task decomposition methods. In each step of the planning, the planner first of all chooses which type of method should be applied[19]. When it backtracks, it can select new methods only with the same type. So in each planning step, the planner can either execute an operator, or apply a decomposition method, or apply a domain refinement method. These options will be referred to as distinct operations.

An algorithm for planning with domain refinement is presented in Figure 7.6. As an input, this algorithm receives a current state of the planner N and initially empty plan P. In each step, the first task in the current task network TA_{Cur}^D is processed. In order to choose an operation that will be applied, the FAF strategy [158] is used, that is the operation with the minimum branching factor is chosen[20]. So the branching factors $K^{Br}(TA_{Cur}^D)$ for operations that can be applied to the first task TA_{Cur}^D are initially estimated. For a compound task, possible operations are task decomposition and domain refinement. When the branching factors for these operations, $K_E^{Br}(TA_{Cur}^D)$ and $K_D^{Br}(TA_{Cur}^D)$, respectively, are both equal to zero for the current task, this indicates a failure, since such task cannot be executed. When both branching factors are equal to a non-zero value, the domain refinement operation takes precedence over the task decomposition operation. This decision is based on the following fact. If a task decomposition method is applied

[19] Otherwise, the search will not be systematic, that is the same solutions can be produced in different search branches that the planner explores when it backtracks. This is due to the fact that the decomposition and domain refinement methods perform different, alternative transformations of the current task network.

[20] The branching factor is equal to number of alternative search branches that the operation produces, that is the number of alternative states that should be explorer in the nest step.

first, one task can be decomposed into several tasks. Hence, further, domains of all new tasks should be refined separately in different planner's steps, increasing the overall number of steps. For a primitive action, domain refinement methods cannot be applied and only operators are used. When the operation that will be executed is chosen, one state from the set of its output values is selected. The postponed policy enforcement procedure is applied for this state, in order to evaluate partial policy requests attached to tasks in its task network (see Section 6.2.4). Only new partial policy requests and partial policy requests for which new information could be added are evaluated. If the postponed policy enforcement procedure is successfully executed, this state is processed further during the planning. When an operator has been executed, the corresponding action is added to the plan structure P.

$DescendingPE$(`State` N,`Plan` P)
1. **If** $tasks(N) = \langle\rangle$ **then Return endif**
2. Assign $TA^D_{Cur} := first(tasks(N))$
3. **If** TA^D_{Cur} is compound task **then** estimate $K^{Br}_D(TA^D_{Cur})$ and $K^{Br}_E(TA^D_{Cur})$:
    3.1 **If** $K^{Br}_D(TA^D_{Cur}) = K^{Br}_E(TA^D_{Cur}) = 0$ **then Return** Failure **endif**
    3.2. **If** $K^{Br}_E(TA^D_{Cur}) = 0$ or $K^{Br}_D(TA^D_{Cur}) \leq K^{Br}_E(TA^D_{Cur})$ **then** Nondeterministically retrieve N_i from $R_D(TA^D_{Cur}, N)$. Execute postponed policy enforcement procedure. **If** Failure is returned **then Return** Failure **endif endif**
    3.3. **If** $K^{Br}_D(TA^D_{Cur}) = 0$ or $K^{Br}_D(TA^D_{Cur}) > K^{Br}_E(TA^D_{Cur})$ **then** Nondeterministically retrieve N_i from $R_E(TA^D_{Cur}, N)$. Execute postponed policy enforcement procedure. **If** Failure is returned **then Return** Failure **endif endif**
    endif
4. **If** TA^D_{Cur} is primitive action **then** estimate $K^{Br}_{Op}(TA^D_{Cur})$:
    4.1 **If** $K^{Br}_{Op}(TA^D_{Cur}) = 0$ **then Return** Failure **endif**
    4.2. Nondeterministically retrieve N_i from $R_{Op}(TA^D_{Cur}, N)$
    4.3. Update P: $P := \langle P, TA^D_{Cur}\rangle$
    4.4. Execute postponed policy enforcement procedure. **If** Failure is returned **then Return** Failure **endif**
    endif
5. **Call** $DescendingPE(N_i, P)$

Figure 7.6: Ordered planning algorithm with domain refinements

A drawback of the descending policy evaluation, which follows from its design, is the existence of additional steps required to select an EP (these steps should be done in order to check if policies in different domains are satisfied). The FAF strategy that we have followed when choosing between domain refinement and task decomposition methods is a universal method that can lead to the decrease of the overall number of planning steps and that can be applied to different types of planners and planning problems. However, in order to apply it efficiently, the planner should have the possibility to select planning operations in each step of the planning from the available alternatives, in a way that the overall number of planning steps is decreased (different variations of this principle were described in [158]). In the previous algorithm, potential gains of this strategy

are limited, since we consider at each planning step only one task, the first task in the current task network. So in order to improve gains of this approach, we designed an extended version of the planning algorithm with domain refinements that can process tasks within the current task network in any order. Additionally, the application of the unordered planning to the BTr development task is motivated by the problem specification mechanism that was defined for the CEP generation problem. Domain constraints in ITr can be specified at different levels of the hierarchy, so they can limit different parts of the track with different extents. However, according to the described principle, decisions which are restricted the most should be made earlier during the planning to reduce the planner's search space. So, as such decisions could refer to slots near to the end of the track, there is a need for a planning algorithm providing the possibility to apply planning operations to different tasks within the current task network in any order during the planning.

In order to apply methods and operators in an unrestricted manner, the following constraints and modifications to the planning domain specification were introduced. When the unordered planning is used, the current planner's world state is not specified fully as operators changing it are carried out non-sequentially during the planning. Accordingly, in order to ensure that preconditions of operators and methods are correctly evaluated, they should refer only to literals with predicate symbols that are not used in operator effects, that is, to the constant part of the planner's world state. When an operator is executed, the operator's effects are not added to the planner's world state directly. They are saved in the planner's world state through specialised literals $Positive(Task_{ID}, p, \tau_1, \ldots, \tau_n)$, where $Task_{ID}$ is the identifier of the action that has been carried out and constructs $p, \tau_1, \ldots, \tau_n$ define the literal used in the effect. The predicate symbol $Positive$ designates that this is a positive effect. Negative effects are added using the same structure with the $Negative$ predicate symbol. When an operator is executed, the action is not deleted from the current task network. Instead, it is added into the set S_{AProc} containing actions that have been already executed and should not be processed further during the planning. Each task in a current task network is assigned an unique identifier $Task_{ID}$ that is used to relate effects saved into the planner's world state with the position within the current task network where they were added. Since we use only fully ordered task networks, it is possible to restore a sequence according to which the effects were executed, in order to build a (partial) planner's world state model for every point of the current task network.

Effects saved using this procedure in the planner world state are unavailable during the precondition evaluation. They are analysed and used only to create (partial) policy requests, when a partial policy vector should be evaluated or when an operator referring to some task within the current task network is carried out. Positive and high-level effects should be added to a policy request if they are assigned to a task before the action for which the partial policy request is created and if they are not clobbered by a negative effect. A positive effect is clobbered by an equal

negative effect assigned to a task after the task referred in the effect and before the task for which the policy request is evaluated. A high-level effect is clobbered by a negative effect that refines it (or is equal to it) and that is assigned to a task after the task referred in the effect and before the task corresponding to the policy request, or it is assigned directly to the effect's task.

If there is a need to specify interrelations between different actions and methods during the planning, like it is done using preconditions and effects, special auxiliary literals are used. These literals are specified using a dedicated set of predicate symbols. These predicate symbols can be used in preconditions and they can be added during the planning. However, they are not related to any specific planner's world state and are not processed during the policy request generation. They are used to manage the decomposition process globally[21]. For example, for the BTr development, literal $used(EP)$ designates that the EP was already used in some task within the current task network, so it cannot be used in other tasks during the planning. In order to define relations between different tasks, in corresponding methods and operators preconditions referring to these literals can be used.

$DescendingPEunord$(`State` N)
1. **If** $\forall TA_i^D \in tasks(N)$ ($TA_i^D \in S_{ADone}$) **then Return** Success **endif**
2. Estimate (update) values of $K_D^{Br}(TA_i^D)$, $K_{Op}^{Br}(TA_i^D)$, $K_E^{Br}(TA_i^D)$ for all $TA_i^D \in tasks(N)$
3. **If** primitive action TA_p^D exists such that $K_{Op}^{Br}(TA_p^D) = 0$ **then Return** Failure **endif**
4. **If** compound task TA_c^D exists such that $K_D^{Br}(TA_c^D) = K_E^{Br}(TA_c^D) = 0$ **then Return** Failure **endif**
5. Choose operation $R_X(TA_i^D, N)$, $TA_i^D \in tasks(N)$ using the following rules:
   - Choose operation with minimum branching factor K_{min}^{Br}, $K_{min}^{Br} \neq 0$
   - If several operations have $K_X^{Br}(TA_i^D) = K_{min}^{Br}$, choose an operation based on operation type priority: first R_D, second R_E, third R_{Op}
   - If several operations with the same type have $K_X^{Br}(TA_i^D) = K_{min}^{Br}$, choose a task that precedes other tasks in task network $tasks(N)$
6. Nondeterministically select N_i from $R_X(TA_i^D, N)$
7. Execute postponed enforcement procedure for N_i. **If** Failure is returned **then Return** Failure **endif**
8. **Call** $DescendingPEunord(N_i)$

Figure 7.7: Unordered version of the planning algorithm with domain refinements

An algorithm for planning with domain refinements when operators and methods can be applied in an order that does not correspond to the order of their application during the plan execution is presented in Figure 7.7. During each step of the planning, all tasks within the current task network are analysed. Branching factors for operations applicable to them are estimated[22]. Similarly as the

[21] Only a part of the planning domain that is used in the BTr development and validation phases was specified in this manner. Moreover, the overlapping mobility scenarios were eliminated.

[22] Not all branching factors are calculated at each iteration. Branching factor values are saved and re-evaluated only when their values can be updated.

previous algorithm, the operation with the minimum branching factor is chosen. If there are several such operations, operation type precedence and operation position precedence are used. Next, the chosen operation is applied. After the operation is applied, the postponed policy enforcement procedure is applied to evaluate partial policy requests attached to tasks within the current task network (only new partial policy requests and partial policy requests for which new information could be added are evaluated). A plan during the unordered planning is not created as a separate entity. Since actions are kept in the task network after they are executed, when the planning is finished, the current task network is analysed and the plan is retrieved from it based on the ordering of tasks. In order to derive time intervals used in partial and fully specified policy requests, a specialised mechanism is used, which extends the mechanism used for the ordered version of the descending policy evaluation. In this mechanism, not only the start time point for an interval depends on the actions that should be executed before the policy request, but also the end time point depends on actions that should be executed after the request.

7.4 Conclusion

The main contribution of this chapter is following. In order to implement the CEP development solution using the problem-independent policy-based planner, described in Chapter 5, the planning environment for this problem area was designed and the CEP generation problem was specified as a planning task in this environment. The specification of this environment includes specification of LObjs, which are used as input and output of the CEP generation process, transformation rules, which are used to transform LObj properties from one scale to another, and the multi-domain hierarchical structure that contains the LObjs and defines the overall structure of the planning environment, interrelations between the LObjs and policies. Importantly, the core processes carried out when a student studies according to a CEP were also specified in this chapter as HTN planning decomposition methods. These processes are utilised during the planning for the CEP development. An additional contribution made in this chapter is the application of the postponed policy enforcement mechanism, described in Chapter 6, to the CEP development planning problem, what resulted in the descending policy evaluation technique development. The descending policy evaluation technique is a problem-specific technique that is aimed at the planning performance improvements within the initial stages of the CEP development process. In concrete, this technique optimises the process of EP intervals selection for the BTr development.

The planning environment specifications described in this chapter should be utilised to solve concrete CEP generation problems. The operation of this planning environment will be considered in Chapter 9, where corresponding case studies are described. The performance gains that can be achieved by the descending policy evaluation technique are also analysed in Chapter 9.

Chapter 8

Implementation

Objectives:

- *Introduce the general architecture and the scope of the prototype.*
- *Provide understanding of the internal organisation of the prototype modules.*

8.1 Introduction

In order to evaluate the techniques presented in the previous chapters, a prototype tool was developed. First of all, this prototype tool implements the domain-independent policy-based planning technique (see Chapter 5) and its extension described in Chapter 6, viz., the postponed policy enforcement mechanism. When this prototype is provided with the tailored planning environment specifications that describe the student mobility problem domain (see Chapter 7), it is able to solve the required task: develop CEPs based on existing EPs and provided requirements.

In order to use the prototype for the CEP development, all necessary information about both the educational environment where the CEP generation problem should be solved and the task that should be solved should be provided. The planning domain specification, which contains methods and operators, specifies the processes carried out within the CEP generation problem area and is stored within the planner in a compiled, unchanging form (see Figure 8.1). Policies that define educational rules and regulations for different domains are specified using the XACML syntax in XML files. They should be saved in a directory where the prototype is configured to retrieve them from. When the prototype is started, it loads policies from this directory, uploads them into its internal registry and waits for the planning problems. A planning problem is provided into the planner as an initial planner's world state specification and an initial task network. The planner's world state contains specification of the educational environment where the CEP generation task should be solved: the domain tree and the specifications of available EPs. CEP property-requirements, ITr and a description of the student are also provided within the initial planner's world state, where they can be easily utilised during the planning (see Figure 8.1). The initial CEP process model is

specified as an initial task network. The planning starts when a file with an initial planner's world state and an initial task network are provided to the system. However, as policies are dynamic and can be changed when the planner is running, the following mechanism for dynamic policy updates was developed. When the policies are updated in the policy directory, names of the policies are also added to a special file. It contains two policy name lists: one for updated policies and another for deleted policies. Each time when the planning is initiated, the prototype checks policy updates in this file and modifies its internal policy repository respectively. When the planning is finished, the planner shows to the user the CEP process models that has been generated during the planning and that solves the specified problem.

The descending policy evaluation technique (see Section 7.3.2) was implemented also as part of the prototype tool. Correspondingly, two versions of the planning domain were developed. The first version supports the ordinary policy-based planning technique only. In the second version, the descending policy evaluation technique and, respectively, the postponed policy enforcement, which it is based on, were implemented. Moreover, the second version of the domain was developed in such a way that both ordered and unordered planning algorithms can be used (see Section 7.3.2). So the prototype can be launched in three modes: first, when the ordinary policy-based planning is used; second, when the descending policy evaluation technique is used with the ordered algorithm; and third, when this technique is used with the unordered planning algorithm.

The architecture of the prototype is presented in Figure 8.1. Components whose main purpose is information processing are represented as rectangles. Components whose main function is data representation are represented as ellipses. Repositories storing data which is rarely modified and which persist between planning sessions are presented as cylinders. The prototype tool includes the following main modules:

- **Planner module** is a component implementing the core planning processes (see Section 8.2). When required, the planner module interacts with the other components. It requests the policy evaluation mediator if policies should be checked during the planning and the transformation rules evaluator if rules should be applied to infer new information. Additionally, the planner module implements functions for interaction with the user using the command line: initiation of the planning problem solving and display of the results.
- **Policy analyser** analyses policies when they are loaded into the prototype and generates the corresponding abstract contexts (see Section 8.4). These abstract contexts are saved into the abstract context repository of the policy evaluation mediator. The abstract contexts are used during the planning for the generation of policy evaluation requests (see Chapter 5). Once the policies are processed by this module, they are registered in the policy repository within the Policy evaluator.
- **Policy evaluation mediator** was developed as a component for the realisation of all interme-

diate processes between the planner module and the policy evaluator (see Section 8.3). The policy evaluation mediator based on policy request vectors provided by the planner module generates XACML policy evaluation requests that are passed to the policy evaluator. When the policy evaluator has evaluated the request, the policy evaluation mediator should process policy evaluation outcomes and provide the final decision concerning the planning process continuation or backtracking. Optionally, if the policy evaluation outcomes contain obligations, the policy evaluation mediator should transform them into an equivalent task network and return to the planner as well. In order to carry out these functions, the policy evaluation mediator uses a registry with abstract contexts generated from policies loaded into the system, has access to the object model of the planner's world state and contains a registry with obligation validation rules.

- **Policy evaluator** (see Section 8.5). The main function of the policy evaluator is the evaluation of XACML policy requests. Its main components are the XACML policy engine and the policy repository. In addition to the standard XACML policy evaluation, this policy engine implements mechanisms for the partial policy evaluation, described in Section 6.3.
- **Transformation rules evaluator** is used to transform notions that are used within the planner's world state to other scales using transformation rules (see Section 8.6). The main components of the transformation rules evaluator are a rules engine and a repository with transformation rules. The rules evaluator is requested by the planner module and the policy evaluator, when they require the rules engine to process the request and infer new values using the transformation rules.
- **Supporting components** are not presented in the figure. They are subsidiary components that are utilised by all core components of the prototype. These components are a logging component and a configuration component. The logging component is used to display log information to the screen or save it to a log file. Using a set of configuration variables, a user can specify which events are displayed to the screen and information about which evens should be saved in the log file. There are 10 different log variables, each of which can specify different log levels for one event type. The configuration component should read from a file and apply a set of settings that determine different aspects of the prototype configuration, for example, a path to the policy directory, the version of the planning algorithm used.

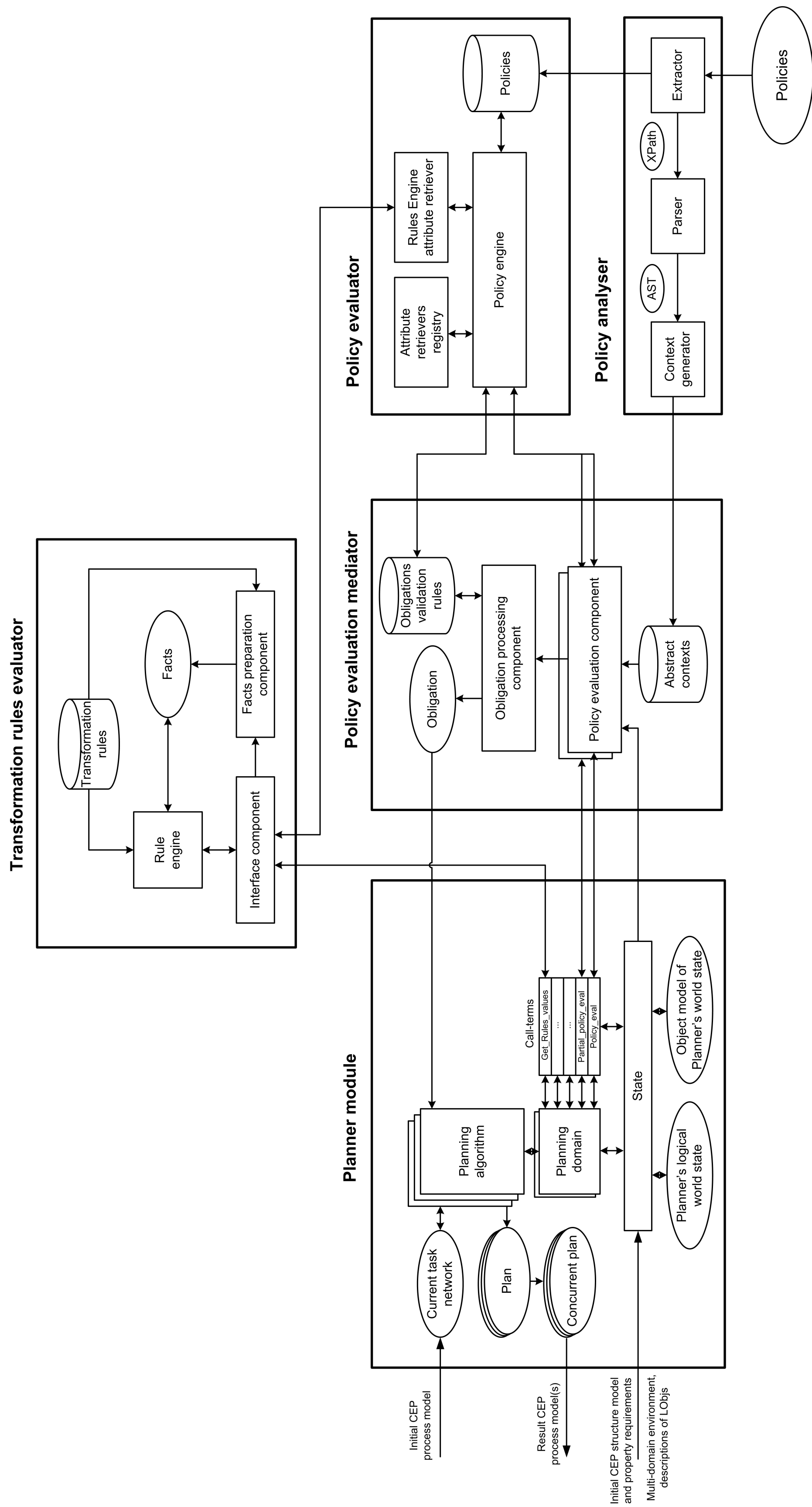

Figure 8.1: Prototype architecture

8.2 Planner module

The planner module was built based on the original SHOP2 planning tool [117], as its extended and modified version. The SHOP2 is an HTN planner that decomposes tasks and applies operators in the same order as they will be carried out during the plan execution. This provides a means to utilise within the method and operator preconditions expressions, which are evaluated against the current planner's world state, for example, universal quantifiers, numeric expressions, call external functions. Currently, there are two implementations of SHOP2, which were developed in University of Maryland: lisp-based version and java-based version, JSHOP2[1]. In terms of functionality these versions are almost equal with the difference that JSHOP2 does not support lisp expressions for comparison and evaluation functions. They are replaced by custom comparison functions and call-terms, which are implemented as java-functions. We use the JSHOP2 implementation in the prototype because this guarantees the smooth integration of the planner into the java-based environment of the prototype. Moreover, as opposed to the lisp version, JSHOP2 is based on domain-compilation approach: a planning domain specification developed should be compiled into java-classes that are invoked during the planning. These java-classes are then used as part of the prototype tool. This approach provides the means to utilise optimisation techniques that can be applied to the domain during the compilation [77].

Based on the original JSHOP2 planning algorithm, three versions of the policy-based planning algorithm were developed for this prototype: the ordinary policy-based planning algorithm (see Chapter 5), the ordered version of the descending policy evaluation planning algorithm and the unordered version of the descending policy evaluation planning algorithm, which can process tasks within the current task network in any order (see Section 7.3.2). In the first version of the planning algorithm, obligations and compound actions support was added to the SHOP2 planner. Additionally, the possibility to transform a linear plan, which is created by the planning algorithm, into a hierarchical plan was added. Operators and methods for the ordinary version of the policy-based planner are specified using JSHOP2 syntax (see [77]). Compound actions decomposition methods were introduced as a distinct class of methods, which unite operators and methods constructs. Expressions for the assignment of time variables, which define action durations, are specified as part of preconditions of the corresponding operators and methods. Policy evaluation initiation and requests to the transformation rules evaluator are implemented using custom call-terms (see Section 8.2.1). The implementation of the planning algorithm with the descending policy evaluation support includes the introduction of different method types (domain refinement and task decomposition), routines for the estimation and storage of branching factor values, production of partially known planner's world states for any task within the current task network (this requires

[1] http://www.cs.umd.edu/projects/shop/description.html [Accessed 22.04.2014]

high-level effects support, increasing and decreasing effects sets generation). Additionally, several packages of custom call-terms were developed, which are used as an elegant tool for extension of the functionality provided by the planner. These functions are used for policy evaluation and rules engine requests initiation, for the implementation of problem-specific functions required during the CEP generation (e.g., a function to check the 'idle time' CEP constraint, a function to check that optional modules selections do not repeat during the planner's backtracks).

8.2.1 Custom call-terms for policy-based planner implementation

Custom call-terms in JSHOP2 provide a means to call custom functions during the evaluation of an operator or method precondition. Using the call-terms, parameters of these functions are passed and their values are returned to the precondition expression (as a value of the call-term). Call-terms are used in the ordinary version of the policy-based planner for two purposes: in order to initiate the evaluation of policies for an action and to carry out a request to the transformation rules engine.

Policy evaluation is initiated using call-term $Policy_eval$ that should be included into preconditions of operators and compound actions decomposition methods. The signature of this call-term corresponds to the policy parameters tuple: $Policy_eval(ObjSet, ParamSet)$. Using this call-term, the planning domain author determines how variables used in the operator or method should be formed into the policy parameters tuple, which provides values for the policy request generation. These variables are used along with corresponding object roles and action parameters names in $ObjSet$ and $ParamSet$ constructs, which are specified as lists within the operator and method schemas (only list structure is supported by JSHOP2). Function corresponding to $Policy_eval$ call-term is called when this term is evaluated during the evaluation of operator's or method's precondition. It receives all designated objects and action parameter values, which are provided to it as call-term parameters. After this, this function determines the action name as the task symbol of the operator or method, from which the function was called, and values $ActBeg$ and $ActEnd$, which it extracts from values assigned to time variables during the evaluation of action duration. Using these values, a fully specified policy vector is built. This vector is passed to the policy evaluation mediator. A Boolean value returned from the mediator is provided as a value of this call-term. If this value is not equal to true, further planning is blocked, as this designates that the precondition is not satisfied.

Transformation rules engine requests are initiated by call-term $getRulesValues$. This call-term should be added into precondition expressions of methods or operators where it is required to utilise a property value in a certain scale, while in the planner's world state this property can be specified in different scales. The following parameters should be provided to this call-term to initiate a request to the rules engine: $getRulesValues(PredicateSymbol, ?Obj_{ID}, scale)$, where

PredicateSymbol is a constant representing predicate symbol used as a name of the property that should be received, $?Obj_{ID}$ is a variable that should be instantiated with an object-term for which the property is being determined, *scale* is a constant representing the scale to which the property of $?Obj_{ID}$, designated using *PredicateSymbol* literal, should be transformed. The function implementing the $getRulesValues$ call-term passes these parameters to the interface component of the rules engine as a request for possible values of the term X in literal *PredicateSymbol*(Obj_{ID}, X, *scale*). The rules engine can derive several values for this term, so call-term $getRulesValues$ always returns a list of values. This list of values should be assigned to a variable in the preconditions expression and processed during the precondition evaluation.

8.2.2 Planner's world state object model

The planner's world state consists of two parts: lower level planner's world state, represented as literals, and higher-level object model of the planner's world state. The object model was implemented as data structures on top of the lower-level planner's world state of JSHOP2 (which is a set of ground positive literals). The class diagram for this part of the prototype is represented in Figure 8.2 and an example of the planner's world state is shown in Figure 8.3. Classes *State* and *TermList* are used to represent literals stored in the ordinary planner's world state of JSHOP2. In JSHOP2, each predicate symbol is mapped to an unique integer identifying it. Therefore, the lower-level planner's world state is contained in object *State* as an array of lists where one list is used to represent all literals with the same predicate symbol. The index of the array is an integer indicating the predicate symbol. Each node in the list contains a sequence of terms representing terms of the corresponding literal. It is modelled using nested structure of *TermList* objects, where each object contains a head term, viz., the current term, and a tail of the list, viz., the rest terms in this list (the tail is represented as another *TermList* object).

Classes *StateGraph*, *StateObject* and *IncidenceElem* were designed to implement the object model of the planner's world state. The object model of the planner's world state is effectively a hyper-graph. We have implemented it using the incidence lists data structure [73]: each vertex-object (i.e., the *StateObject* object) contains a list of references to incidence edge-objects. This data structure was chosen based on the following premises: first of all, it provides the possibility to represent hyper-edges; secondly, it is a structure that fits for storage of sparse graphs that the object model of the planner's world state belongs to; thirdly, it is appropriate for effective implementation of common operations with it (i.e., addition/deletion of edges, retrieval of edges incidence to the current vertex); moreover, in this structure, edges and vertices are represented as distinct objects so it is possible to reuse existing objects representing logical terms (i.e., *TermList* objects) as edges without the need to duplicate information. For storage of vertices representing objects within the object model hypergraph of the planner's world state, an array $Objects$ is used. It is contained in

StateGraph class, which is a subclass of *State* class. The index of the array identifies the object within the object model of the planner's world state: it is equal to the identifier of the object-term representing this object (each term in JSHOP2 is mapped to an unique integer). Each object-vertex is represented as *StateObject* object. Within this object, three separate lists for different types of incidence edges are stored: *BinPropList* for binary properties, *PropList* for other properties and *RelList* for relations edges. This provides the possibility to process these edges separately. Each edge in the lists is represented using an *IncidenceElem* object containing an integer identifying the predicate symbol of the literal and a reference to the *TermList* object with the sequence of literal's terms. Therefore, in the designed ad-hoc structure for the planner's world state object model, only the required minimum information is stored. It includes information about object vertices and their relations within the object model hypergraph. Information about property vertices and edges themselves is stored using the ordinary JSHOP2 objects.

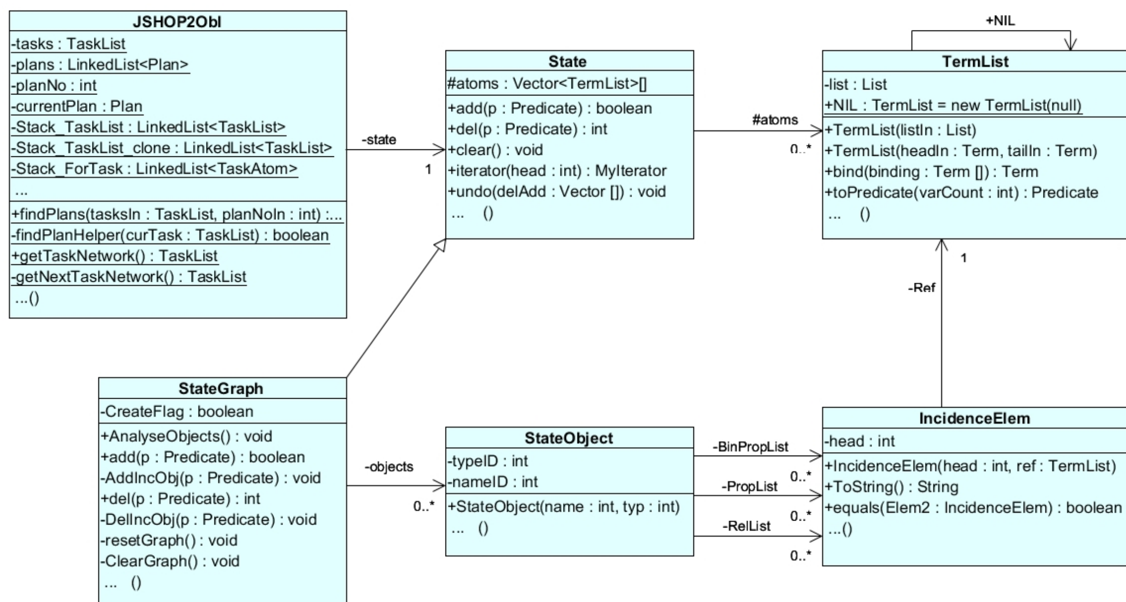

Figure 8.2: Class diagram for the planner's world state

8.3 Policy evaluation mediator

The policy evaluation mediator was developed to support interactions between the planner module and the policy evaluator and carry out transformations of data structures from the JSHOP2 object model to the XACML object model and back. In order to perform these functions, this module stores abstract contexts and obligations validation rules in corresponding repositories (see Figure 8.1). A class diagram for the policy evaluation mediator is represented in Figure 8.4. The policy evaluation mediator carries out the following main functions:

Policy requests generation. The policy evaluation mediator is called when a policy evaluation

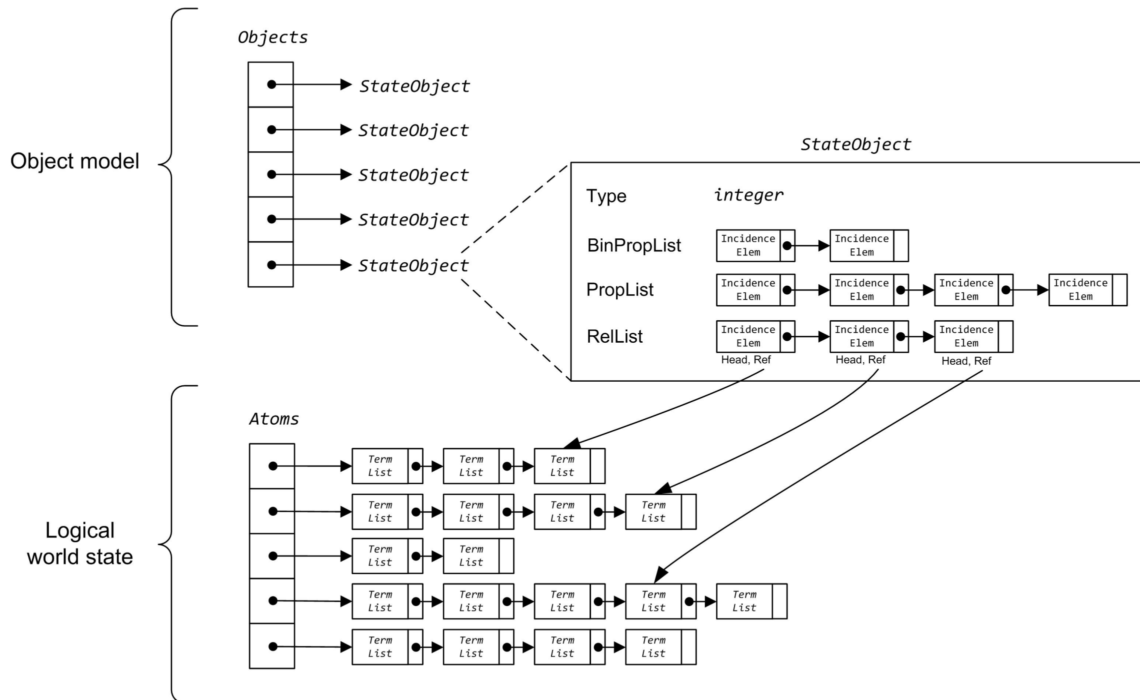

Figure 8.3: Example of the planner's world state

call-term is processed during the planning: the policy request vector, containing all required parameters for the policy request generation, is provided from the planner and the policy evaluation mediator starts the policy request generation using the algorithm described in Chapter 5. When generating a request, it uses information stored in the object model of the planner's world state and the abstract contexts registry. The policy request definition in the policy-based planner extends the standard XACML policy request, so the following conventions are utilised to construct the policy request. Contexts for designated objects are stored within the *tree* element as its first level children. In order to distinguish designated objects with different roles, a role of designated object is used as a prefix for all nodes within its context. Attribute values for designated objects distinct from objects with '*Subject*' or '*Resource*' roles are stored as resource attributes. In order to distinguish attributes of different objects, a concatenation of a role of the designated object and an attribute identifier is used as an identifier of this attribute. Corresponding conventions are also used for the specification of policies. Additionally, since attribute values in XACML are stored and processed in a typed form, when a policy request is generated, data types of attributes should be determined. Data types for object attributes are fixed for a pair containing an attribute identifier $Attr_{ID}$ and an object role. For every designated object, obligatory attributes *type*, *id* and *role* have *String* data types. For action attributes, data types are fixed for a pair containing an action name and an attribute name. Obligations and conditions used in the policy-based planner

are specified as extensions of ordinary obligations in the XACML policy language. After a request is generated, it is passed to the policy the evaluator for the evaluation. In the class diagram of the policy evaluation mediator (see Figure 8.4), the *PolicyEvaluation* class is responsible for this functionality.

Policy evaluation outcomes processing. After a policy decision request is evaluated by the policy evaluator, the policy evaluation decision is interpreted by the policy evaluation mediator according to the rules defined in Chapter 5 and the outcome (a Boolean value) is returned to the planner module. Obligations returned by the policy evaluator are processed and transformed from the XACML object model into an internal data structure: information about all obligations and conditions attached to one policy decision is represented by one *ActionOutcomes* object (see Figure 8.4). For both conditions and obligations, three lists are contained in the *ActionOutcomes* object: a before, during and after list. Conditions are stored as *String* objects in the corresponding lists. Before, after and during lists of an obligation are represented as *OblList* objects. This object can contain ordered and unordered lists and supports transformation of stored obligations into corresponding task networks. Each obligation in a list is contained as an *OblTaskProcessed* object. After all obligations returned by the policy evaluator are transformed into the internal object model, they are validated against the obligations validation rules, which were specified for the action being evaluated at the planning domain level. This is carried out by the *ObligationValidRules* object. If the obligations are successfully validated, the *ActionOutcomes* object is returned to the planner module for further processing. In order to transform a list of obligations from the internal object model to the JSHOP2 object model, the *OblList* object has function *generateTaskNetwork* that performs this transformation and returns JSHOP2 *TaskList* object containing an equivalent task network. During this transformation, atomic tasks are generated by the *generateTask* method of the corresponding *OblActionProcessed* objects.

Obligations validation rules retrieval and storage. Obligation validation rules are specified in a file using the language a grammar for which was specified in Chapter 5. When the system is launched, this file is parsed and objects corresponding to obligation validation rules objects are created. The *ObligationValidRules* object is used as a registry for obligation validation rules at both the planning domain and policy levels. The *ObligationValidRules* class has methods for the obligation validation rules parsing and for the validation of obligations. The obligations validation rules registry is also requested by the policy evaluator when it requires validating intermediate obligations at the policy level.

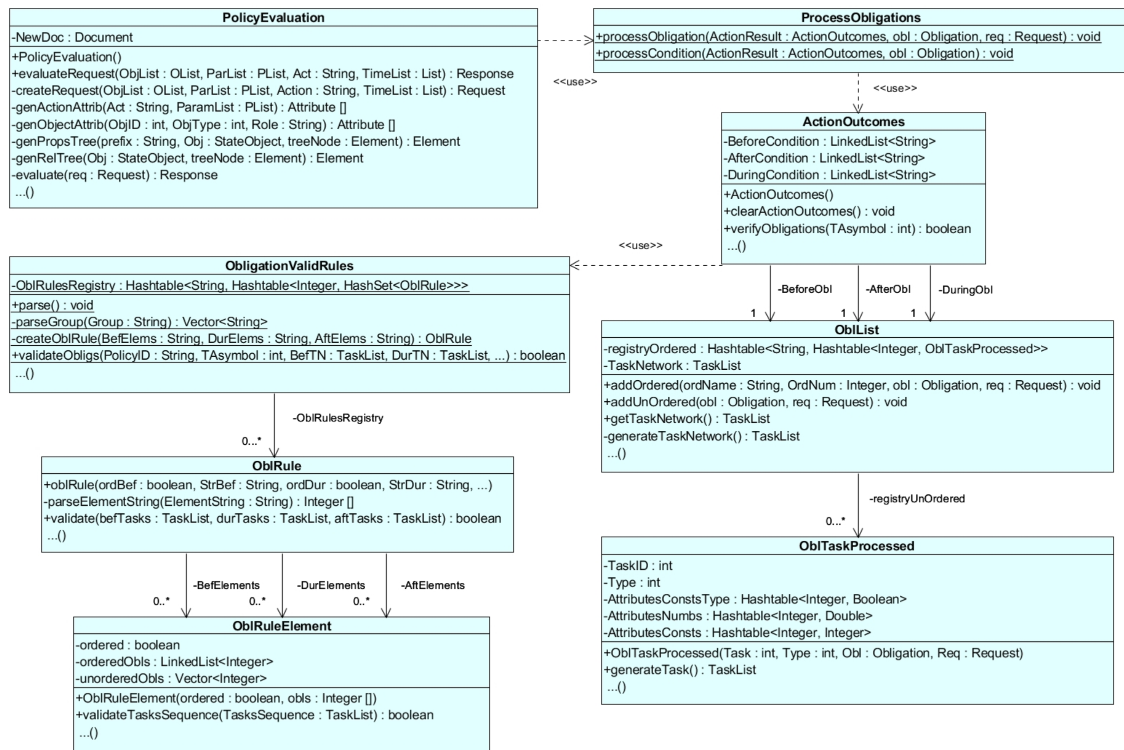

Figure 8.4: Class diagram for the policy evaluation mediator

8.4 Policy analyser

In order to generate a policy decision request that contains all relevant information from the planner's world state, the abstract contexts mechanism was introduced in Chapter 5. Abstract contexts are used to specify which information can be required for the evaluation of XPath expressions contained in policies. The policy analyser was developed in order to analyse policies being loaded into the system and generate abstract contexts for them. The policy analyser consists of three blocks that sequentially process XPath expressions used in policies: extractor searches and extracts XPath elements from *Attribute Selector* elements of XACML policies; parser parses and verifies these XPath expressions and builds Abstract Syntax Trees (ASTs); and context generator transforms an AST into a set of abstract context trees, that are used to update the abstract contexts registry.

In order to implement the parser, we used the ANTLR[2] parser generator that generates $LL(*)$ parsers based on context-free grammars. This gives the possibility to specify our grammar without limitations on lookahead. Using this tool, we generate a lexer and parser from the lexer and parser grammar rules that are specified within a single grammar file using the unified ANTLR syntax. For this purpose, we have adopted the ANTLR grammar for the XPath language[3], which fully conforms

[2]http://www.antlr.org/ [Accessed 22.04.2014]
[3]http://blog.jwbroek.com/2010/07/antlr-grammar-for-parsing-xpath-10.html [Accessed 21.04.2014]

to XPath language specification v.1.0 [33]. We have introduced the following modifications to this grammar: first, we removed syntax constructs that are not supported by the abstract context generator[4]; second, we have added to this grammar AST generation constructs. Additionally, we have split some right hand-side expressions for parser rules into several sub-expressions in order to have the possibility to specify distinct generative parts for each sub-expression. ANTLR provides convenient tools for the specification of AST generation rules in-line with main parsing grammar rules: operators and rewrite rules. Rewrite rules are more expressive. When they are added to the rules of the parsing grammar, a full-fledged generative grammar is specified.

We have used ASTs as intermediate data structures between XPath expressions and abstract context trees in order to eliminate XPath elements that are not used for the abstract contexts generation and to represent the XPath expression as a tree structure, which provides the basis for the abstract context trees generation. A basic element of an XPath expression is a location path, which selects a set of nodes relative to a current set of context nodes[5]. A location path is a sequence of steps, separated by '/' symbols. Each step is an expression that defines how new context nodes can be determined based on the current context node. An AST fragment generated from one location path has the following form: each step in a path is represented by one node '$STEP$', which is a child of a node '$ABSPATH$' if it is an absolute path or '$RELPATH$' if it is a relative path (an absolute path starts with '/' symbol, a relative path starts with a step element). Parsing/rewrite rules 2 - 4 in Figure 8.5 were specified to generate such AST structures during the XPath parsing. Generative part of a rewrite rule is specified after the '->' sign (see rules 1 and 4.)[6]. This part is added to the AST when the corresponding parsing rule is applied during the XPath parsing. If the rewrite part is not specified, all constructs in the right-hand side of the parting rule are added to the AST next to the node that was added last (see rules 2 and 3). As can be seen, if any information is not used for the abstract context construction, it is omitted in the AST[7]. For this purpose, redundant elements are omitted in the generative part of the rewrite rule or, if a rewrite is not specified for the parsing rule, they are marked by operator '!' in the right-hand side of the parsing rule.

'$STEP$' nodes have child nodes representing constructs that are used in the XPath expression to specify how new context nodes should be selected. Thus, a '$STEP$' node may have child constructs representing step's axis, a node test expression (full qualified node name or a pattern

[4]Only child, attribute and parent axes are supported. Non-abbreviated syntax for axes is eliminated. As follows, each element within the XML document that can be utilised during the XPath evaluation should be explicitly referred in the XPath expression.

[5]At any time point during the XPath expression evaluation, there is a set of nodes called (current) context nodes. Each expression (and sub-expression) being evaluated should be applied to each node in this set. As a result of the expression execution, the current node in the context set is substituted with new nodes produced by the expression.

[6]When '^(...)' construct is utilised, the first element in the brackets is used as a sub-root of the AST, other elements are its children.

[7]For example, they are specific symbols (e.g., colons in prefixed qualified names, steps separators in location paths, brackets) or constructs representing whole entities (e.g., comments, text nodes, variable references).

to select a node based on its name and type) and a predicate (see rule 4 in Figure 8.5). The axis is used to define how new context nodes are related with the current context node. The node test expression is used to select nodes based on their names and types. Finally, predicate part is used to specify additional conditions that new context nodes should satisfy. By default, 'child' axis is used for all nodes, so this axis is not represented in the AST. When axes '.' and '..' are used, they are marked in the AST, since they change the way how next context nodes are determined relatively to the current context node and this should be considered during the abstract context construction (see rules 4 and 5). Attribute axis '@' is also not represented in the AST since abstract contexts does not specify restrictions on XML attributes (steps with this axis are not eliminated during the AST construction since they can contain predicate parts that specify restrictions on XML elements). Rules 6 and 7 parse 'nodeTest' construct of the step, which defines a pattern that the XML entity should satisfy for its usage as a new context node. In abstract contexts, only XML elements are represented, so other XML entities, for example, processing instructions and comments (they are represented as '**NodeTypeOthers**' token), are ignored. When no restrictions on some part of the element name are defined in the XPath expression (e.g., placeholders '*' or 'node ()' (represented as '**NodeTypeNode**' token) are used), specialised node '**ANY**' is added into the abstract context. Specified parts of the element name (they are represented as '**NCName**' token or 'qName' non-terminal) are added as AST nodes.

1. `locationPath` : `relativeLocationPath` ->^(**RELPATH** `relativeLocationPath`)
| `absoluteLocationPathNoroot`->^(**ABSATH** `absoluteLocationPathNoroot`);

2. `absoluteLocationPathNoroot` : '/'! `relativeLocationPath`;

3.`relativeLocationPath` : `step` ('/'! `step`)*;

4. `step` : `nodeTest` `predicate`* ->^(**STEP** `nodeTest` `predicate`*)
| `abbreviatedStep` ->^(**STEP** `abbreviatedStep`)
| '@'! `nodeTest`! `predicate`* ->^(**STEP** `predicate`*);

5. `abbreviatedStep` : '.' ->^(**SELF**) | '..' ->^(**GOUP**);

6. `nodeTest` : `nameTest` | **NodeTypeNode** '(' ')' ->^(**ANY**)
| **NodeTypeOthers**! '('! ')'! | 'processing-instruction'! '('! **Literal**! ')'!;

7. `nameTest` : ' * ' ->^(**ANY**) | **NCName** ' : '! ' * ' ->**NCNameANY** | `qName`;

Figure 8.5: Parsing/rewrite rules for processing location paths and generation of AST

In XPath there are points where location paths can be nested or branched, meaning that more than one step should be applied to the context node during the XPath expression evaluation. Nesting of location paths can happen in filter expressions and in steps' predicates. These parts of the expression can contain separate location paths. In AST, a nested location path is represented as a subtree whose root is added as a child node to the '*STEP*' node representing location step in the predicate or filter expression. XPath constructs that can lead to branching are logical connectives

(and, or), set operations (union), comparison operations (equal, less, more), and others. These constructs can contain several location paths as operands. When such operations are processed, the AST should be branched: a new node representing the operation is added and each lower level location path is added as its child (see examples of rules for processing of these situations in Figure 8.6).

1. Disjunction: `orExpr` : E = `andExpr` ('or' F = `andExpr`)* ->^(**OR** $\$E$ $\$F^*$);

2. Conjunction: `andExpr` : G = `equalityExpr`
('and' H = `equalityExpr`)* ->^ (**AND** $\$G$ $\$H^*$);

3. Equality: `equalityExpr` : A = `relationalExpr`
((' = '|'! = ') B = `relationalExpr`)* ->^ (**EQUAL** $\$A$ $\$B^*$);

4. Relations: `relationalExpr` : C = `additiveExpr`
((' < ' | ' > ' | ' ≤ ' | ' ≥ ') D = `additiveExpr`)* ->^ (**REL** $\$C$ $\$D^*$);

5. Arithmetic operations: `additiveExpr` : I = `multiplicativeExpr` (('-' | '+') J = `multiplicativeExpr`)* ->^ (**ADD** $\$I$ $\$J^*$);

Figure 8.6: Examples of parsing/rewrite rules that introduce AST branching

The complete grammar for XPath parsing and AST generation is represented in Appendix C[8]. An example of XPath expression, retrieving information from the context of a designated object with role 'Subject', and an AST that was generated for it are presented in Figure 8.7[9]. This XPath retrieves names of modules related using any relation with a subject who knows English language at the level of '4'. This XPath request contains two predicate parts: for steps *Sub* : *student* and *Sub* : *lang*. At the points of AST corresponding to these predicate parts, the AST is extended with subtrees with *OR* root nodes. Nodes representing relative location paths contained in these predicates parts were generated within these subtrees. This XPath expression contains three operators that lead to AST branching: two equalities and one conjunction.

The algorithm for the abstract context generation based on AST is represented in Figure 8.8. The procedure *AddPath* initiates the abstract context tree generation: it finds the first step and generates the abstract context tree root. The procedure *STEPproc* processes *STEP* nodes of the AST representing the location path in a recursive manner and generates the rest of the abstract context tree. When location paths represented in an AST are converted into abstract context trees, each location path is transformed into a path in the tree. All *STEP* nodes that are children of the same *RELPATH* / *ABSPATH* are transformed into a simple path. Each *STEP* node is mapped to a vertex if the previous step was transformed into an edge or to an edge otherwise (see lines 2 and 3 in the procedure *STEPproc*). *STEP* nodes are processed from left to right in the

[8]It should be noted that examples of rules from this grammar described in this section were adapted to facilitate the readability of the grammar rules, in concrete, tokens were defined implicitly in rules' bodies.

[9]Within an XML document representing a designated object context, the role of this object in the policy request is specified in prefixes of XML nodes.

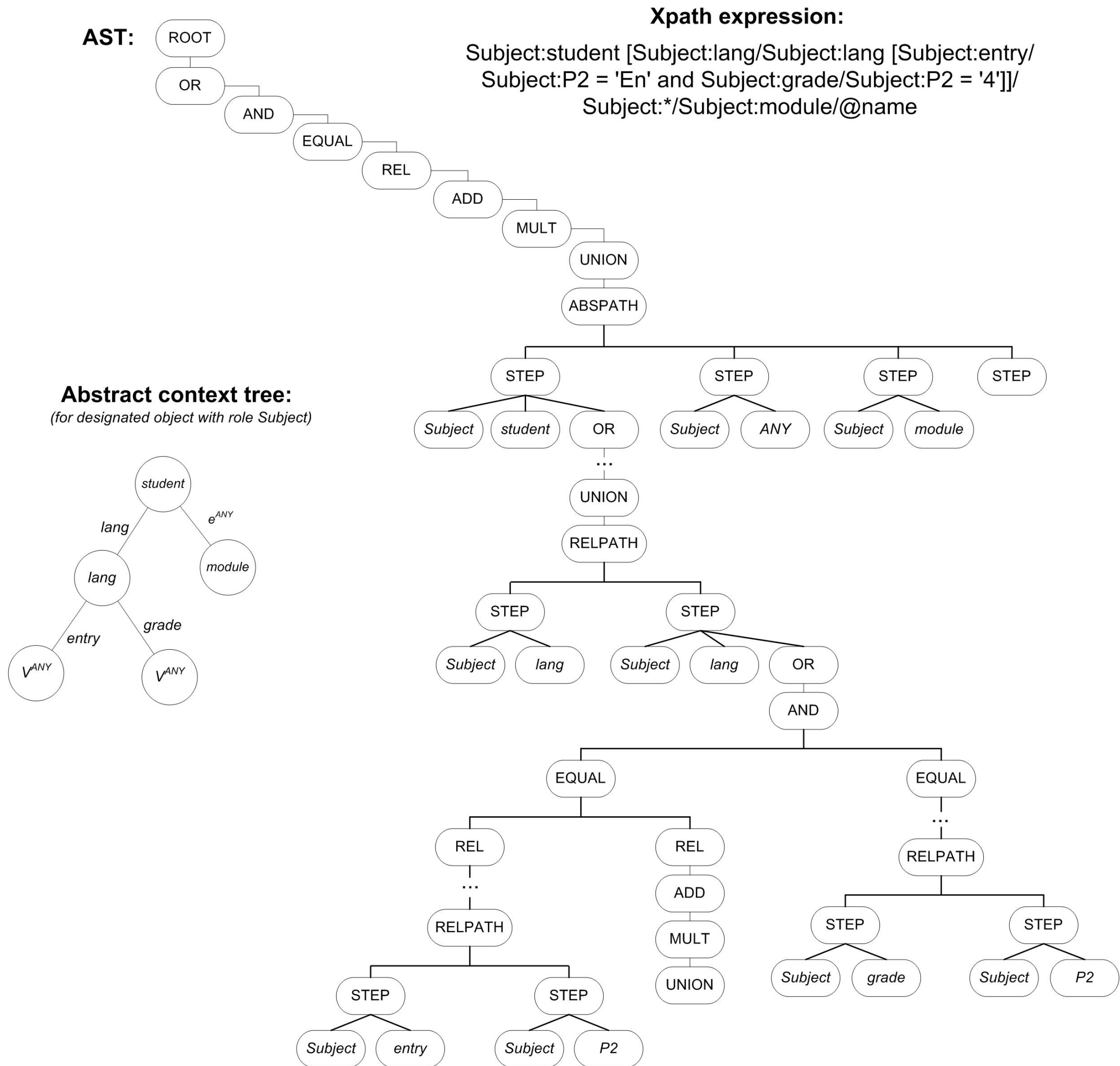

Figure 8.7: XPath expression, corresponding AST and abstract context tree

AST and new elements are added in the abstract context tree from the root vertex to leaves[10]. When a new absolute location path is found, a new abstract context tree is created (see line 4.3 in the procedure $STEPproc$). When a new relative location path is found within a subtree where the current $STEP$ node is used as a root, a new branch for this location path is added to the current abstract context tree (see line 4.4). This new branch is added to a vertex or an edge representing the current $STEP$ node. If this $STEP$ node was transformed into a vertex, the first $STEP$ node of new relative location path is added as a new edge for this vertex. If this $STEP$ node was transformed into an edge, this edge is cloned and the first $STEP$ node of the new relative location

[10]When AST $STEP$ nodes are transformed into abstract context vertices, nodes representing a property of an object are transformed into V^{ANY} vertex, since abstract contexts are not used to filter objects based on their properties (AST nodes representing object properties are detected using their names: they satisfy the 'PN' pattern, where N is a position of the property in the literal).

path is added as a vertex incident with the clone of this edge. After one abstract context tree is generated, it should be checked that all its edges and vertices have the same prefix (or some edges and vertices can have no prefix) (see line 4 in the procedure *AddPath*). $GOUP$ steps are processed specially. They move the current element in the abstract context tree being generated one element higher: to the previous vertex or edge. It should be noted, when the evaluation of an XPath expression used as a top-level expression in an Attribute Selector element is started, the *AddPath* procedure is utilised, supporting both absolute and relative top-level XPath expressions. This is possible since the current context node in this situation is equal to the root of the XML document being processed.

When the abstract context tree is built, it is added to the abstract contexts registry within the policy evaluation mediator. As was described in Chapter 5, abstract contexts are organised using the leading variables mechanism. Each combination of values of leading variables and a role of the designated object is mapped to a set of abstract context trees. In the prototype, we used one leading variable: the domain variable. The abstract contexts registry is constructed using *Hashtable* objects for mappings and *HashSet* objects for storage of abstract context trees. The mapping $Domain \times Role \rightarrow \{VertexElement\}$ is stored in order to map a domain value and a role value to a set of abstract context trees that were generated from XPath expressions retrieving information from contexts of designated objects with this role used in policies for this domain ($VertexElement$ objects contain root vertices of abstract context trees).

All policies specified for a domain are also in force for all its descendant domains. Therefore, when the ordinary version of the policy-based planning is used, an abstract context tree generated is 'merged' with a tree set for the current domain and tree sets for all lower level domains[11]. The 'merge' operation is used since extracted abstract context tree can repeat some part of an abstract context tree that already is in the tree set. In this case, only distinct part of the tree should be added. Correspondingly, a new tree is added to the tree set only if its root vertex is not mergeable with a root vertex of any tree in the set. Rules for the identification of mergeable vertices and edges are described in Chapter 5.

8.5 Policy evaluator

The core component of the policy evaluator is the XACML policy engine. This engine carries out the evaluation of policies for the XACML policy evaluation requests provided. In the prototype, we have adopted the Enterprise Java XACML 2.0 policy engine[12], which fully supports XACML 2.0 specification. We have extended this policy engine in order to introduce mechanisms for the partial policy evaluation (see Section 6.3). This XACML policy engine has the following

[11] During the descending policy evaluation, separate policy requests are generated for each domain, so this is not required.

[12] http://code.google.com/p/enterprise-java-xacml/ [Accessed 23.04.2014]

Inputs: $CurNode$ - the first ABSPATH/RELPATH node found during the left depth-first tree traversal of AST.

Procedure $AddPath$ ($CurNode$)
1. $CurNode$:= first left $STEP$ child node for $CurNode$
2. Construct abstract context vertex V based on child nodes of $CurNode$, $CurElement$:= V
3. **If** $CurNode$ has more $STEP$ nodes next to the right **then**:
   3.1. $CurNode$:= next to the right $STEP$ node for $CurNode$
   3.2. **Call** $STEPproc$ (‘$vertex$’, $CurElement$, $CurNode$)
   endif
4. **If** not all prefixes within the tree with root $CurElement$ are the same or absent **then** Raise an exception **endif**
5. Add the tree with root $CurElement$ to the abstract context registry

Procedure $STEPproc$ ($CurType$, $CurElement$, $CurNode$)
1. **If** first child of $CurNode$ is $GOUP$ **then**:
   1.1. **If** $CurType = \text{‘}edge'$ **then** $CurType := \text{‘}vertex'$ **else** $CurType := \text{‘}edge'$ **endif**
   1.2. $NewElement$:= element (vertex or edge) of abstract context tree that is incident with $CurElement$ and is closer to the root
   1.3. $CurNode$:= next node to the left from $CurNode$ in AST
   1.4. **Call** $STEPproc$ ($CurType$, $CurElement$, $CurNode$)
   endif
2. **If** first child of $CurNode$ is not $SELF$ and $CurType$ = ‘$edge$’ **then**:
   2.1. Build $NewElement$ vertex from child nodes of $CurNode$, add it to the abstract context tree and make incident with $CurElement$
   2.2. $CurType$:= ‘$vertex$’
   endif
3. **If** first child of $CurNode$ is not $SELF$ and $CurType$ = ‘$vertex$’ **then**:
   3.1. Build $NewElement$ edge from child nodes of $CurNode$, add it to abstract context tree and make incident with $CurElement$
   3.2. $CurType$:= ‘$edge$’
   endif
4. **If** there is OR node in child nodes of $CurNode$ **then Loop** for each OR:
   4.1. **If** $NewElement$ is not instantiated **then** $NewElement := V^{ANY}$ **endif**
   4.2. $FoundPath$:= first $ABSPATH$ or $RELPATH$ node found during left depth-first tree traversal of subtree with the OR root node
   4.3. **If** $FoundPath$ = $ABSPATH$ **then Call** $AddPath$ ($FoundPath$) **endif**
   4.4. **If** $FoundPath$ = $RELPATH$ **then**:
      4.4.1. **If** $CurType$ = ‘$edge$’ **then** $Element$:= a copy of $NewElement$ **else** $Element$:= $NewElement$ **endif**
      4.4.2. $NewNode$:= first $STEP$ node of $FoundPath$ subtree
      4.4.3. **Call** $STEPproc$ ($CurType$, $Element$, $NewNode$)
   Endloop endif
5. **If** $CurNode$ does not have $STEP$ nodes next to the right **then Return endif**
6. $CurNode$:= next to the right $STEP$ node from $CurNode$
7. **Call** $STEPproc$ ($CurType$, $NewElement$, $CurNode$)

Figure 8.8: Algorithm for Abstract context generation from AST

advantages, in comparison with other XACML engines. It has cache mechanisms, which improve the performance of the policy evaluation. The policy cache facilitates retrieval of policies applicable to policy requests being processed. The evaluation results cache saves policy evaluation outcomes

for the processed policy requests. During backtracking, the planner module can generate repeating policy requests, so this feature is highly required (although for the performance experiments in Chapter 9 this functionality was disabled). Other useful features of this engine are extensibility and integrability of its architecture. Next, it is described how these possibilities of the XACML policy engine are utilised in the prototype.

The policy evaluator was integrated with the transformation rules evaluator using the mechanism of pluggable *attribute retrievers*, provided by this policy engine. A rules engine attribute retriever was developed which is called in order to form a request to the rules evaluator. In the policy specification, the specially introduced Attribute Designators construct is used, which designates that it is required to retrieve attribute values in specific scale. This Attribute Designator, in addition to the attribute identifier, contains the scale parameter $Scale$, which is equal to a scale in which the value should be returned. This Attribute Designator element is evaluated during the evaluation of the corresponding policy. When its attribute identifier is equal to a predicate symbol contained in the list of predicate symbols supported for transformation, the rules engine attribute retriever is called. It creates a request $\langle property, Obj_{ID}, Scale\rangle$ to the transformation rules engine. The attribute name specified in the Attribute Designator is used as the property name $property$. The designated object Obj_{ID} for which the attribute is being retrieved is determined based on its role, which is also specified in the Attribute Designator. Another possibility to call the rules engine attribute retriever is during the evaluation of an Attribute Selector with an XPath expression. When it is detected that the XPath expression retrieves a property of an object and the predicate symbol of the property-literal is contained in the list of predicate symbols supported for transformation, the rules engine attribute retriever is called. In this case, the $Scale$ parameter for the request should be specified in the predicate part of the XPath step retrieving the property value.

Moreover, using the extensibility of this policy engine, as part of the partial policy evaluation implementation, the new combining algorithms and evaluation components, supporting the Indeterminate temporal decision and other new values, were developed and registered in the policy engine instead of standard algorithms. Finally, new functions that can be used in policy conditions were implemented using *functions provider* functionality of the policy engine. `type-bag-sum(Bag)` function calculates the sum of all numbers in the *Bag*. `modules_sim_max( Modules_bag, Module, Thrs)` function is used for the comparison of modules based on the maximum similarity measure (see Section 7.2.2). It returns `True`, if the module *module* is similar with one of the modules contained in the modules set *Modules_bag* by more than threshold *Thrs*[13].

[13]Comparison of modules is based on known values of the modules similarity measure $sim^{mod}(Mod_1, Mod_2)$.

8.6 Transformation rules evaluator

The transformation rules evaluator is requested by the planner module and the policy evaluator when they need to infer new values during the policy evaluation or during the precondition evaluation using rules specified by the domain author. For the rules engine, we have adopted tuProlog[14], a java-based Prolog engine, which readily integrates into the java environment of the prototype. The rules evaluator also contains the rules repository, which stores the prolog rules (see Figure 8.1). The interface component of the rules evaluator is used to support interactions with the requesting modules: generate prolog queries using information provided, retrieve required information from the planner's world state and extract result values from the evaluation outcomes.

8.7 Conclusion

This chapter describes the prototype developed to test the techniques designed in the previous chapters. The scope of the implementation, the process of the prototype usage and its overall environment were described. The general architecture of the prototype and descriptions of each of its main components were provided. Several core techniques for the policy-based planner implementation were described in more detail, in concrete, the abstract contexts generation technique, which involves the policy analysis and XPath expressions parsing, and the implementation of the object model for the planner's world state. The main contributions of this chapter are the development of the general architecture of the prototype implementing the CEP generation framework and a more detailed design of its components. This prototype will be used in Chapter 9 for the evaluation of the solution proposed in this thesis.

[14]http://alice.unibo.it/xwiki/bin/view/Tuprolog/ [Accessed 08.07.2011]

Chapter 9

Evaluation

Objectives:

- *Show using case studies the feasibility of the CEP generation solution designed.*
- *Evaluate the required properties of the policy-based planner, which is used as a core engine in the CEP development solution.*

9.1 Introduction

In this chapter, the described research is evaluated and its practical applicability is shown. Two case studies, described in Section 9.2, illustrate the practical applicability of the presented research for the CEP development support. In these case studies, policy-based planning is used to solve a planning problem where a set of possible CEPs satisfying the user's requirements should be developed based on existing EPs. In Section 9.3, we analyse properties of policy-based planning that were specified as input requirements for its development. It was required that the planning should be carried out in environments with heterogeneous regulations managed by several persons independently. In Section 9.4, several series of experiments are described that were carried out to analyse the performance of the policy-based planner in planning environments with different characteristics and evaluate the performance gains produced by different versions of the descending policy evaluation technique: the version with the ordered planning and the version with the unordered planning.

9.2 Case studies

9.2.1 Case study 1

In case study one, a medium scale planning environment in considered. In order to create a sufficiently large planning environment, a domain tree is built using fictitious domains and EPs. This makes the planning problem easily scalable, as analogous EPs and policies can be used. In order to solve the CEP generation planning problem in this environment, we have applied the

approach described in Section 7.3.1.5. In this approach, the planner initially develops a set of BTrs which are provided to the user and he (or she) has a possibility to update the problem statement or choose a concrete BTr for the further CEP development. In order to demonstrate the different possibilities for the BTr development, in addition to the ordinary policy-based planning (see Chapter 5), the ordered and unordered versions of the descending policy evaluation technique are used, which were designed to improve the performance of the BTr development phase when large-scale planning problems are considered (see Chapter 7).

In the scenario for this case study, a CEP with two permanent transfers should be developed for a student (i.e., when the student does not return to EPs where he (or she) has studied already). Moreover, this CEP should contain only partner transfers (when receiving and home universities are partners) according to the established partner network (see Chapter 3).

Planning environment. The domain hierarchy for this case study is presented in Figure 9.1. There are 2 countries, 5 universities and 15 EPs in this domain. All these EPs are unified: they have 4 semesters (6 months duration each) and lead to identical awards, BSc in Computer science. For the aims of this case study, only high-level descriptions of the EPs are provided (without modules specifications). Within these EP descriptions, it is specified which languages are used for teaching. The policies of the universities in this domain require that students know all languages from this set in order to study an EP.

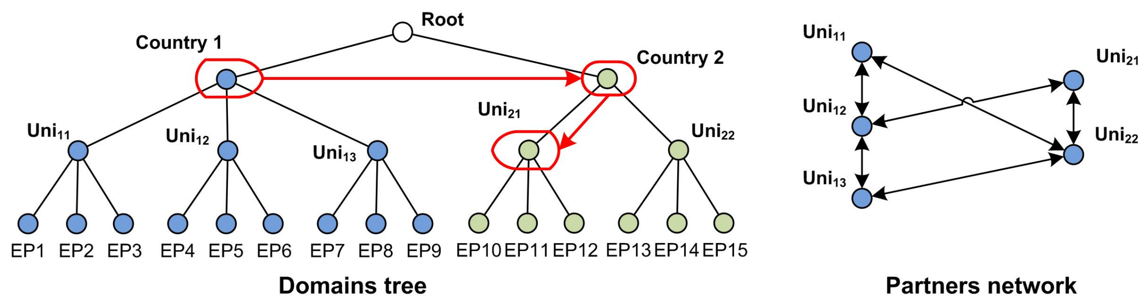

Figure 9.1: Domain tree schema (case study 1)

Policies. Examples of schemas for $Country_1$ and Uni_{11} policies are presented in Figure 9.2. Policy sets corresponding to different domains are specified in different files, so they are shown separately in the figure. Lower level policy sets and policies are specified using nested structures. XACML *AttributeDesignator* elements are designated like ordinary variables in the figure. For example, the '*in*' attribute designator in policy set targets: this attribute designator is applied to a resource designated object and returns a domain where this object (i.e., the corresponding LObj) is contained. As was specified in Section 7.2.3, domains and LObjs form a domain tree, which is a hierarchy of properties, hence equality relations for them are evaluated considering their positions in the current domain tree (see Chapter 6). Therefore, a policy set is applicable to all LObjs

included into a domain which is used in the equality relation in its target. *AttributeSelector*s, which select values from the policy request using XPath expressions, are designated as *Selector* [⟨*Value description*⟩], where expressions in square brackets describe values retrieved using the *AttributeSelector*'s XPath. For example, *Selector* [*EP_language*] attribute selector is used in a condition of rule in the policy P_6 that specifies requirements to language level of a student. It returns a set of languages that are required for the education at the EP where the student wants to study.

In this case study, analogous policies are specified for all domains at the same level. The country policy sets (for example, PS_2) contain only policy sets with language constraints, consisting of two policies. The policy for international students P_2 contains a rule that such students in order to study in this country should know one of the languages that can be used for teaching in this country, at the required level[1]. For native students, the separate policy is specified where language requirements are omitted (it is assumed that such students know the required languages). The university policy sets contain the following policies. Firstly, an university admission policy P_5 prescribes that only students with certificates from specific countries can be admitted for studying at BSc programmes. Next, the policy set PS_4 defines a partner network for the university using the following rules. If a student transfers from another university, he (or she) can be admitted to the university only if the previous university is within a specified set of partner universities. The policy P_3 within this policy set defines inner-universities transfers rules: these transfers are permitted only for native students[2]. In addition to the mentioned earlier policy for EP language requirements (P_6), the university policy P_7 imposes limitations on durations of EP intervals that can be studied in the university. If a student transfers to the university for a probation period (in this case, the action $tranfer_IT$ is used to designate a temporal transfer), the duration of study should be less than 7 months. If a student transfers to a university to get a degree (in this case, the action $tranfer_IP$ is used to designate a permanent transfer), he (or she) should study at the EP that he (or she) will graduate from for a period longer than 10 months.

Policies in different domains differ in constants used in their condition parts. Policy constants for all policies are presented in Table 9.1. For example, a partner network policy for the university Uni_{21} can be read as "in case of inter-university transfer, the student can transfer to Uni_{21} only from Uni_{22} or Uni_{12}". The partner network, defined by these policies, is presented in Figure 9.1. It also should be noted that this case study considers only BTr development, so in all policy requests the student is used as as an designated object with the role 'subject' and the EP interval that this action refers to is used as an designated object with the role 'resource'.

[1] Language levels are specified and compared according to the CEFR scale.

[2] That is, they are permitted for students from the country where the corresponding university is situated.

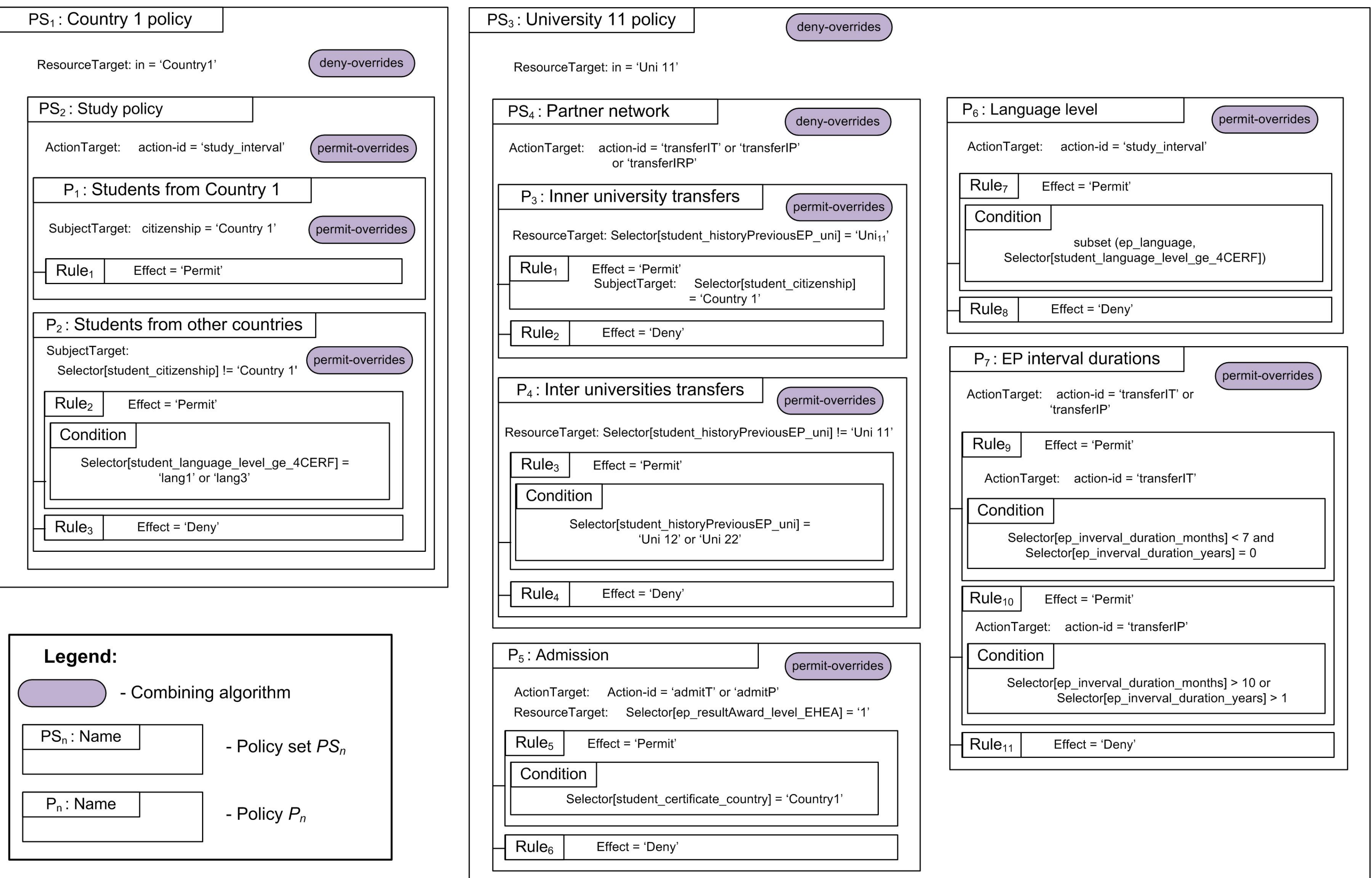

Figure 9.2: Schemas for $Country_1$ and Uni_{11} policies (case study 1)

Table 9.1: Specification of constants, used in policy rules for different policies (case study 1)

Policies	**Domains** / Constants				
PS_2 Study policy (P_2 Foreign students)	**For Country$_1$**			**For Country$_2$**	
	$Lang_1$, $Lang_3$			$Lang_2$, $Lang_3$, $Lang_4$	
PS_4 Partner network (P_4 Inter transfer)	**For Uni$_{11}$**	**For Uni$_{12}$**	**For Uni$_{13}$**	**For Uni$_{21}$**	**For Uni$_{22}$**
	Uni_{12} Uni_{22}	Uni_{11}, Uni_{13} Uni_{21}	Uni_{12} Uni_{22}	Uni_{12} Uni_{22}	Uni_{11}, Uni_{13} Uni_{21}
P_5 Admission	$Country_1$	$Country_1$ $Country_2$	$Country_1$ $Country_2$	$Country_2$	$Country_2$
P_6 Language level (EP requirements)	EP_1, EP_2 : $Lang_1$ EP_3 : $Lang_1$, $Lang_3$,	EP_4, EP_5 : $Lang_1$ EP_6 : $Lang_3$	EP_7 : $Lang_3$ EP_8, EP_9 : $Lang_1$	EP_{10}, EP_{11} : $Lang_2$ EP_{12} : $Lang_3$, $Lang_4$,	EP_{13} : $Lang_2$ EP_{14}, EP_{15} : $Lang_3$, $Lang_4$

Problem statement. It is required that a CEP with three slots and two transfers is created for student $Student_1$. The higher-level task $Degree(Student_1, Track_1, Award_{req})$ is used in the problem definition as the initial task. The ITr specified as $Track_1 = \langle Country_1, Country_2, Uni_{21} \rangle$ (see Figure 9.1) determines sets of EPs that can be used in each of the three slots of this CEP. The student should start the education in $Country_1$, then transfer to $Country_2$ and, finally, make a transfer within $Country_2$ in order to graduate from university Uni_{21}. So the student must make two permanent transfers during the education. A fragment of the initial planner's world state, which is used in this problem, is presented in Figure 9.3. In this problem, only a minimum amount of information necessary for this case study is specified. It is defined that $Student_1$ is from $Country_1$ and has a certificate from this country. He (or she) knows languages $Lang_1$ and $Lang_2$ at levels 6 and 4 respectively, according to the CEFR. His (or her) goal that should be achieved using the CEP developed is a degree in the area of Computing according to the ISCED taxonomy with a level equivalent to the first cycle of the EHEA QF issued by an education provider in $Country_2$. As is shown in Figure 9.3, values of properties that can be specified using different scales are specified along with their scales, as was described in Chapter 7. This enables the conversion of these properties using transformation rules. Additionally, the time constraints for the resulting CEP are specified within the initial planner's world state: minimum CEP start date ($t_{Beg} = \langle 01, 09, 2011 \rangle$), maximum CEP end date ($t_{End} = \langle 15, 09, 2013 \rangle$) and a maximum time interval between adjoining EP intervals in the CEP ($\delta^t = 3$).

Course of planning. In order to solve the problem, we first use the original version of the policy-based planning. When the planner is started, abstract contexts are created based on *Selector* elements of the policies given. As all policies are homogeneous, abstract contexts for different domains on the same level are equal. Abstract contexts generated based on the country and university policies are presented in Figure 9.4. According to the country's context, only information about the student's language qualification and citizenship should be added into the

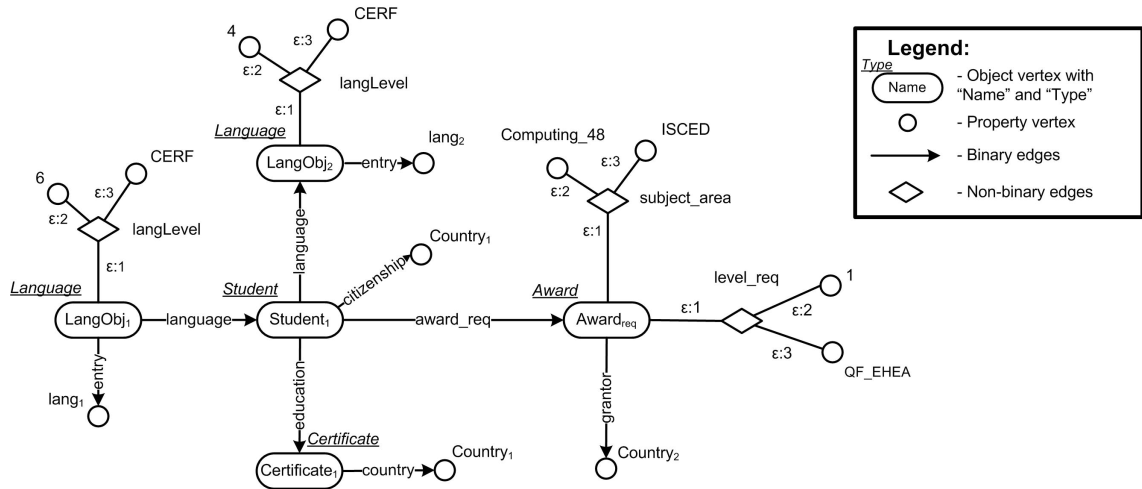

Figure 9.3: Initial planner's world state (case study 1)

context. According to the university's abstract contexts, in addition to the student's language qualification and citizenship, information about his (or her) previous education should be added: both about EP intervals studied within the CEP being developed and pre-HE certificates, that are analysed when the student is admitted to a university. In addition, for the EP used as a designated object with the 'Resource' role, information about the award level and durations of possible EP intervals should be added. Since policies for a specific domain are in force for all its descendant domains, abstract contexts derived for a domain are merged with contexts for all its ascendant domains. In this case study, abstract contexts for EPs are absent and contexts for countries are included in the contexts for universities. Correspondingly, the abstract context that is used during the generation of a policy request where an EP is used with the 'resource' role is equal to the abstract context for the university of this EP.

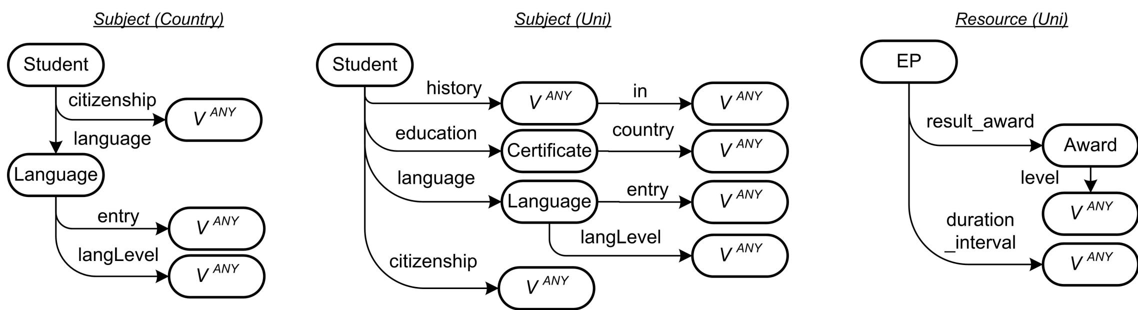

Figure 9.4: Abstract contexts for university and country policies (case study 1)

The task networks decomposition structure that the planner has generated before the first CEP is found is presented in Figure 9.6. Numbers in circles designate the order of the decomposition execution. The initial task $Degree(Student_1, Track_1, Award_{req})$ can be decomposed only using

method represented in Formula 7.3.1.2 that implements two permanent transfers student mobility scenario. One permanent transfer student mobility scenario (see Formula 7.15) cannot be applied, since it is not applicable to three-slot tracks. The probation period student mobility scenario (see Formula 7.3.1.2) is not applicable, since the same EP cannot be used in the first and the last slots. Therefore, when the planning starts, using the method in Formula 7.3.1.2, the *Degree* task covering three slots is decomposed into three one-slot tasks. The EP interval $|EP_1|_{(1,1)}$ is chosen within $Country_1$ for the *Start_degree* task (when the planner backtracks, other applicable EP intervals from $Country_1$ will be tried for the *Start_degree* task). Next, the tasks generated are decomposed in the order according to which they will be executed during the education: first, the task *Start_degree* is decomposed into three actions and they are immediately executed[3]. During the execution of actions, policy requests are generated and evaluated. These policy requests are evaluated against policies for the $Country_1$ and Uni_{11} domains. Example of the policy request for the &*study_interval* compound action, which is generated using the described abstract context, is presented in Figure 9.5[4]. According the abstract context, information about student's language skills is added into the context of the student object[5]. Relying on his (or her) level of $lang_1$ language, $Country_1$ policy returns a *Permit* decision. During the policy evaluation for the Uni_{11} domain, his (or her) language skills are checked against the EP_1's language requirements, which are represented using *EP_language* attribute of the resource designated object (it is retrieved using an attribute designator in the policy). These requirements are also satisfied. Policy decisions generated by the domain policy sets are combined using $deny - overrides$ policy combining algorithm and finally a *Permit* decision is returned by the policy engine. Other information added into the policy request contexts is required by policies for other actions, for example, information about student's certificates and the educational level of the EP that the student studies is used during the admission policy evaluation. Information about studied EP intervals (represented using *history* predicate) is absent in the planner's world state, as the considered action &*study_interval* is carried out within the first slot of the track. Information about EP interval durations for the EP used as the resource designated object is represented in literals with predicate symbol '*duration_interval*'[6]. It is utilised by the EP interval durations policy of the university. In the described manner, all policy requests generated for actions that decompose the *Start_degree* task are evaluated. They all are permitted and corresponding operators are successfully executed.

Next, the task *Proceed_degree* is decomposed into actions. During this decomposition, the EP

[3] &*choose_modules* compound action is not used in this case study, as at the level of the BTr development optional modules are not selected within EP intervals used in the BTr.

[4] The policy request is shown without the time parameters.

[5] It is used in the request as designated object with the 'Subject' role.

[6] The first, second and third terms in this literal specify the EP interval (they contain the identifier of EP object and numbers of the start and end semesters of the considered EP interval). The fourth and fifth terms specify duration of the considered EP interval, that is, the number of years and months. The last term contains the duration in days.

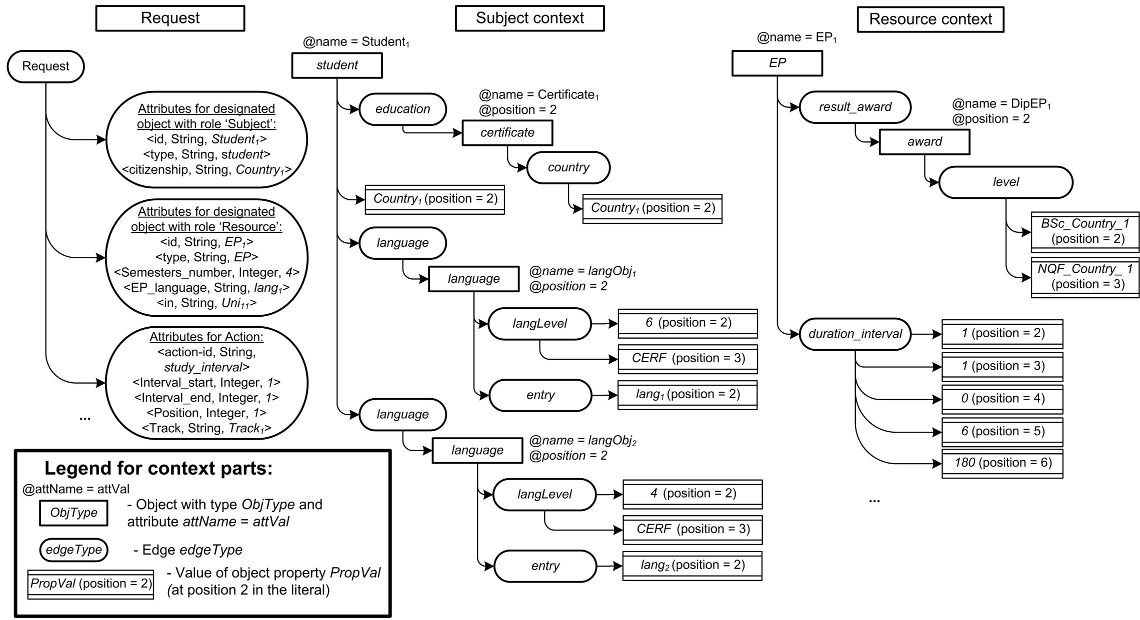

Figure 9.5: Policy request for $\&study_interval$ action. Original policy-based planning.

interval that will be used in the second slot is chosen within $Country_2$. For each EP in the country, only two EP intervals ($|EP|_{(2,2)}$ and $|EP|_{(2,3)}$) are considered, because the planner checks that the time interval between the adjoining EP intervals does not exceed the specified limit. In our case study, it is three months so after the last odd semester in $|EP_1|_{(1,1)}$ only an even semester can be the first semester in slot 2. First, EP intervals for EP_{10}, EP_{11} and EP_{12} are sequentially tried, but they all are denied during the evaluation of policies for the action $\&tranfer_IT$. These EPs belong to Uni_{21} that does not have partner relations with Uni_{11}, where the student has studied during the first slot. In Figure 9.6, numbers of dead-end decompositions are shown inside red circles. For each rejected EP interval, two decompositions have been spent: the decomposition of $Proceed_degree$, when an EP interval is chosen, and the operator execution for the $\&transfer_IT$ action, when it is recognised that this branch is illegitimate. After these dead-ends, the planner tries the $|EP_{13}|_{(2,2)}$ interval, which is from the Uni_{22} university and which can be used in this slot according to the partner network policy of the university Uni_{11}. Other policy checks for this EP return positive or N/A results as well: conditions in the EP interval duration policy are satisfied, as duration of this interval is 6 months, the language level policy returns 'Permit' since the student knows $Lang_2$. Finally, the task $Finish_degree$ is processed in the similar manner. An EP interval $|EP_{10}|_{(3,4)}$ is tried first. All actions generated with this EP interval are legitimate: the EP interval duration is one year (this is enough for the interval duration policy for the $\&transfer_IP$ action) and $Student_1$ knows language $Lang_2$ (this is required for studying EP_{10} according to the language level policy). After the execution of the $!graduate$ action, the first BTr is found: $\langle |EP_1|_{(1,1)}, |EP_{13}|_{(2,2)}, |EP_{10}|_{(3,4)} \rangle$.

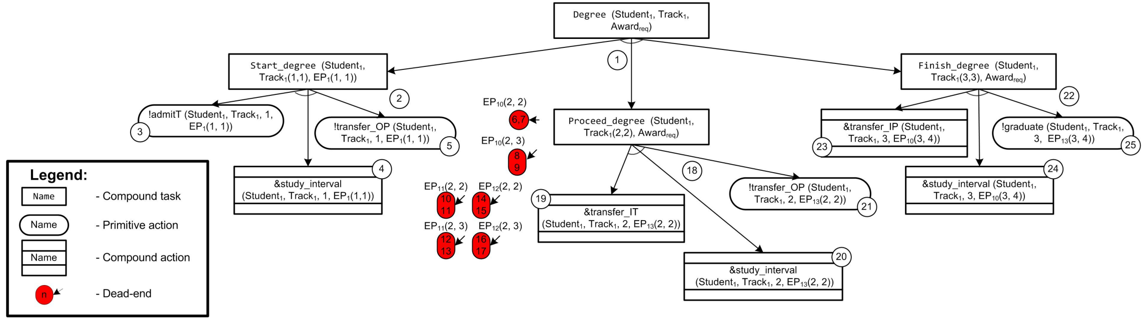

Figure 9.6: Task network decompositions structure. Original policy-based planning.

When the first BTr is found, the planner starts backtracking: from the last produced tasks to the first task. For each task, it searches for other possible EP intervals. This case study was designed in a way that in each slot only one EP interval for each EP can be used: other BTrs are incorrect because of the EP interval duration policy, or the overall CEP duration constraint[7] or the limit on time intervals between slots (i.e., 'idle time'). From the partner network diagram it is clear that possible sequences of universities where the student can study are $\langle Uni_{11}, Uni_{22}, Uni_{21}\rangle$ and $\langle Uni_{13}, Uni_{22}, Uni_{21}\rangle$. There are no sequences starting from $\langle Uni_{12}, Uni_{21}\rangle$, because $Student_1$ is from $Country_1$ and inner-university transfers are denied for international students by the policy of Uni_{21} university. The EP language requirements limit the number of EPs that can be used in each university. In the first slot, the student can study only EP_1 and EP_2 in Uni_{11} and EP_8 and EP_9 in Uni_{13}. In the second slot, only EP_{13} is legitimate, and in the third slot - EP_{10} and EP_{11} in Uni_{21}. All 8 BTrs where different combinations of these EPs are used were generated by the planner.

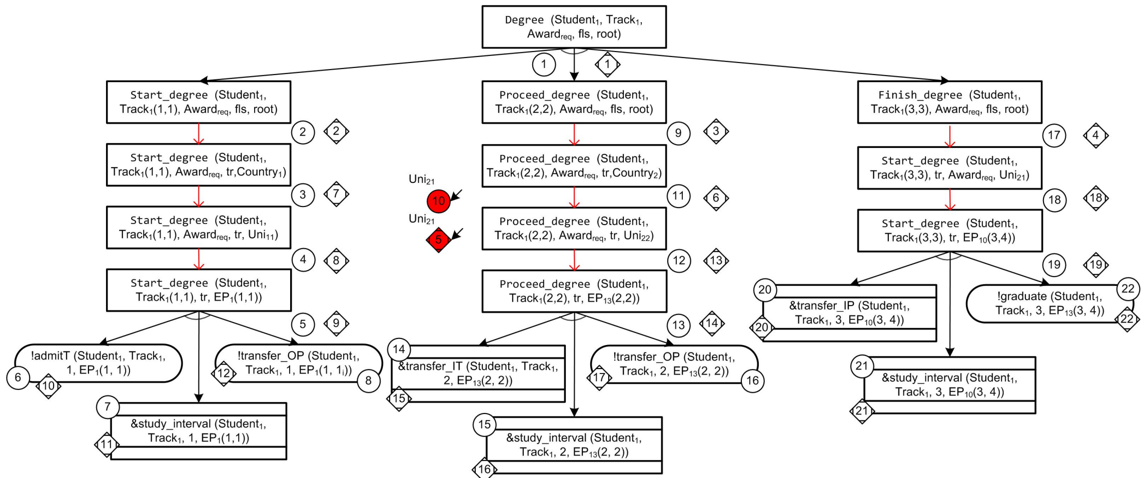

Figure 9.7: Task network decompositions structure. Descending policy evaluation.

[7] Duration of actions, which update time variables during the planning, are equal to durations of the corresponding EP intervals.

When the descending policy evaluation technique is used, abstract contexts are not merged with contexts for the descendant domains, because policy requests are evaluated separately for each domain. Hence, abstract contexts constructed for country policies and university policies (see Figure 9.4) are used separately, for the generation of policy requests at the country and university level. Since abstract contexts for EPs are absent, the designated object contexts for policy requests at this level are not created. The task network decompositions structure generated during the planning before the first BTr is found is presented in Figure 9.7. The ordered version of descending policy evaluation decomposes tasks from left to right, in the same order as tasks are executed during the education. The order of decompositions execution is shown using numbers in circles. Domain refinement methods are designated as red arrows, decomposition methods - as black ones. The first decomposition is the same as in the previous version[8] with the difference that for all three generated tasks EP intervals are not selected. Instead, the root domain is used as the current domain for these tasks. In addition, during this decomposition high-level effects and partial policy vectors are created and attached to the generated tasks. One high-level effect $history(Student_1, Track_1, Slot_num, Dum_EP_i, NIL, NIL)$ is attached to each $Start_degree$, $Proceed_degree$ and $Finish_degree$ tasks. It shows that $Student_1$ in slot $Slot_num$ studies EP Dum_EP_i. Dum_EP_i is a dummy EP created for this slot and representing all known properties of the EP which will be used studied in this slot. These properties include the currently known domain of the EP (country or university), the level and area of its award, which are known from the student's requirements (in our case it is a degree in Computing with a level equivalent to the first cycle of the EHEA QF). High-level effects with predicates *passed_exams* and *admitted* are attached to the $Start_degree$ task and one with *education* predicate is attached to the $Finish_degree$ task. Additionally, after the first decomposition, each task has three partial policy vectors attached, representing 3 actions that will be executed when this task is fully decomposed. Each partial policy request uses $Student_1$ object in the subject role and the corresponding Dum_EP_i object in the resource role. Action parameters in these partial policy requests contain only slot number. Time intervals of these requests are loose, they represent possible action execution time computed based on the CEP start and end time limits. These partial policy requests are not evaluated at this stage of the planning, because they refer to the root domain, which does not contain policies.

The ordered descending policy evaluation algorithm first chooses for processing the task $Start_degree$. Only domain refinement methods are applicable to this task at this stage, so the planner refines the domain for this task in the following sequence: $root \rightarrow Country_1 \rightarrow Uni_{11} \rightarrow |EP_1|_{(1,1)}$ and updates the domain of the dummy EP used in the corresponding high-level effect and partial policy vectors. In the second step, it applies the domain constraint specified in the corresponding slot of the ITr ($Country_1$ domain) and sets Enf^{ITr} flag to true. During each re-

[8] Similarly with the original policy-based planning, other methods cannot be applied to the *Degree* task.

finement of the domain, the planner evaluates all three partial policy requests in a new domain. The policy request for the action &*study_interval* in the domain $Counrty_1$ is presented in Figure 9.8. This policy request uses $DummyEP_1$ as a resource. Only *in* attribute is specified for it, because other binary object properties for this dummy EP are not known at this stage. Context information about this object is not required according to the abstract context at the county level. The context for the student object, which is used as a resource designated object, contains only information about the language levels and citizenship, because other available information is not required according to the corresponding abstract context. This partial policy request is evaluated to *Permit*, because it contains all available information for evaluation of the policy set for the $Country_1$ domain. Partial policy requests for !*admitT* and !*transfer_OP* are evaluated to N/A, because country policy sets do not contain corresponding policies. Partial policy requests for the Uni_{11} domain can be derived from the country policy requests if the domain of $DummyEP_1$ is updated to Uni_{11} and information about student's certificate is added into the subject's context and information about $DummyEP_1$ level is added into the resource's context, according to the university abstract contexts in Figure 9.4. So the partial policy request for !*admitT* is evaluated to *Permit* according to the Uni_{11} policies, because this request indicates that the student has a certificate from the $Country_1$ domain and an EP that will be used in this action will have a level equivalent to the first cycle of the EHEA QF[9]. Partial policy requests for the !*transfer_OP* and &*study_interval* actions are evaluated to N/A and *IndTemp* respectively. A permanent decision for the latter request cannot be inferred as the request does not specify languages required for studying the EP. These requirements should be checked during the evaluation of the language level policy (partial policy evaluation mechanism detects that language requirements can be added to the resource context further, as a dummy EP is used). During the final domain refinement, the EP interval is chosen for the *Start_degree* task. $DummyEP_1$ is substituted with real EP_1 and, hence, a fully specified policy request can be generated. Policy requests generated for EP_1 return N/A, since corresponding policies are not specified. However, after their evaluation, postponed policy request for the &*study_interval* action is re-generated and re-evaluated for the Uni_{11} policy. This policy request becomes equal to the request in Figure 9.5 and is evaluated to *Permit*. After these domain refinements, the task *Start_degree* is decomposed into actions !*admitT*, &*study_interval* and !*transfer_OP*. During their execution, high-level effects are changed to ordinary effects. Policies for them are not evaluated since all required evaluations were carried out for higher-level tasks.

Next, domain refinements are applied to the *Proceed_degree* task. The domain constraint from the ITr is applied and the domain is changed to $Country_2$. Evaluation of partial policy

[9]Since a dummy EP is used as a resource, new information can be added about it but available information about its level is sufficient to infer a permanent decision by this policy.

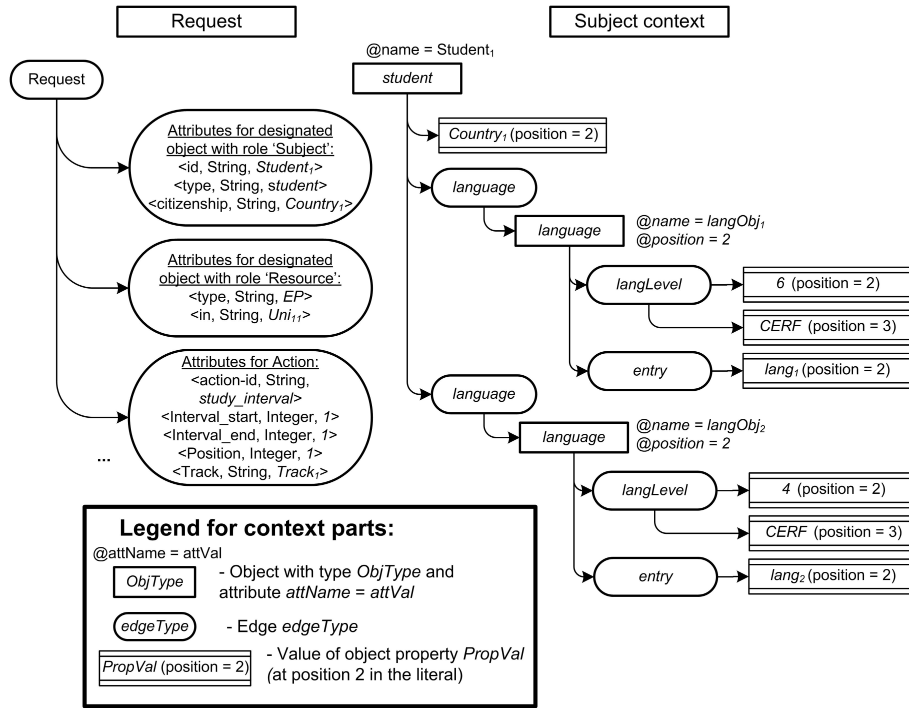

Figure 9.8: Partial policy request for $\&study_interval$ action in $Country_1$ domain. Descending policy evaluation.

requests for $Country_2$ is identical to evaluation of requests for the previous task at the country level (request with $\&study_interval$ action is evaluated to $Permit$, other requests - to N/A). After this, the university is chosen. First, Uni_{21} is tried, but then the partial policy request with the $\&transfer_IT$ action is evaluated to $Deny$, because of the partner network policy. During backtracking, the next university, Uni_{22}, is tried. The partner network policy for this university permits the transfer, but the interval duration policy returns $IndTerm$, because the dummy EP is used and durations of its semesters are not specified. These decisions are combined into the $IndTemp$ decision, indicating that action $\&transfer_IT$ should be re-evaluated further. When the interval $|EP_{13}|_{(2,2)}$ is chosen for this task, it is evaluated as $Permit$. The planning proceeds for the $Proceed_degree$ and $Finish_degree$ tasks without dead-ends until the first BTr is found. Partial policy requests for actions $\&study_intervals$ and $\&transfer_IP$ are evaluated to $IndTemp$ at the university level, because the EP language constraints and intervals durations are not known then. However, when this information is available, they are permitted. Other policy requests are evaluated to N/A. It should be noted that the domain constraint in the ITr is specified for the $Finish_degree$ task at the university level. So, when this constraint is applied, policies are evaluated for university Uni_{21} and country domain $Country_2$ in one step, therefore, only two domain refinements are required for this task. When the first BTr is found, the planner starts backtracking. When a domain refinement operation is backtracked, the next possible EP interval

or domain is tried, in order to develop new BTrs. So the planner generates all 8 BTrs, which were created by the previous version of the planning algorithm.

The task networks decomposition structure generated when the unordered version of descending policy evaluation is used is the same as the structure for the ordered version. The difference is in the order according to which the tasks are processed. The numbers of steps for the unordered version are shown in diamonds in Figure 9.7. This version of the planning algorithm uses the branching factor minimisation principle to select a task for further processing. Minimum branching factors for tasks *Start_degree*, *Proceed_degree* and *Finish_degree* at different steps of the planning, when they are specified for different domains, are shown in Table 9.2. Branching factors estimated before the application of ITr domain constrains are equal to one (only one domain refinement method can be applied). Therefore, tasks with the root domain are decomposed one immediately after the other. After these domain refinements, partial policy requests are evaluated in new domains: for the first and the second slots they are $Country_1$ and $Country_2$. For these requests $Permit$ and N/A decisions are returned. For the third slot, Uni_{21} is used, so policy requests for $\&transfer_IP$ and $\&study_interval$ are evaluated to $IndTemp$ because of the partner network and language policies. After this step, the minimum branching factors are equal to the number of branches generated using the domain refinement methods, that is, to the number of children for the current domain. So the task *Proceed_degree*, having the minimum K_{Br} (equal to 2), is refined next at step 5. First, Uni_{21} is chosen and a dead-end is detected. But it is not the action $\&transfer_IT$ at slot 2 that is evaluated to *Deny*. Indeed, the partial policy request for this action is evaluated to $IndTemp$, because university in slot 1 is unknown in this step. However, after the evaluation of the policy requests for the *Proceed_degree* task, postponed partial policy requests for other tasks are re-evaluated. The partial policy request for the action $\&transfer_IP$, attached to the *Finish_degree* task, is evaluated to *Deny*, because, based on the universities chosen, an inner-university transfer is detected, which is prohibited for international students by the university policy. So the last domain refinement is backtracked and the next university Uni_{22} is tried for the task *Proceed_degree*. All policy requests after this refinement are evaluated to $IndTemp$ and N/A, so the planner successfully goes to the next iteration where it processes the *Start_degree* task, for which the $Country_1$ domain is refined into Uni_{11}. For partial policy requests attached to the *Start_degree* task, all policies are evaluated to $Permit$, $IndTemp$ and N/A. After their evaluation, postponed policy requests for *Proceed_degree* and *Finish_degree* are re-evaluated (however, decisions for these requests are not updated). When the current domain for a task is a university, the minimum branching factor for such task is equal to the number of possible EP intervals, which can be used in this task. Hence, branching factor increases when a university is chosen as a task's domain. In this case study, each university has the same number of EPs, so such tasks have equal branching factors (equal to 9). Therefore, EP intervals for tasks within the current task network are chosen in

the order according to which these tasks will be executed during the education. When an EP interval is selected, the minimum branching factor is equal to number of branches that can be generated using decomposition methods (because domain refinement cannot be applied to such tasks anymore). Only one decomposition method, producing one branch, can be applied to such tasks, so they are decomposed immediately and primitive actions produced are immediately carried out (they also have branching factors equal to one). Therefore, primitive actions are carried out in the same order as in the ordered version. Additionally, when an EP interval for one task is selected, branching factors for other tasks where EP intervals are to be selected are re-evaluated and reduced based on the information about the already chosen EP interval (see lines 9 - 14 in Table 9.2). All policy requests generated for tasks with known EP intervals are fully known and permanent decisions are always returned for them. Using the unordered version of descending policy evaluation, the same set of BTrs as when using the previous versions is generated.

Table 9.2: Minimum branching factors for tasks estimated before planning step execution

Step	*Start_dgr*	*Proceed_dgr*	*Finish_dgr*	Step	*Start_dgr*	*Proceed_dgr*	*Finish_dgr*
2.	*Root* - 1	*Root* - 1	*Root* - 1	13.	-	Uni_{22} - 6	Uni_{21} - 9
3.	$Country_1$ - 3	*Root* - 1	*Root* - 1	14.	-	EP_{13} - 1	Uni_{21} - 3
4.	$Country_1$ - 3	$Country_2$ - 2	*Root* - 1	19.	-	-	Uni_{21} - 3
5.	$Country_1$ - 3	$Country_2$ - 2	Uni_{21} - 9	20.	-	-	EP_{10} - 1
7.	$Country_1$ - 3	Uni_{22} - 9	Uni_{21} - 9				
8.	Uni_{11} - 9	Uni_{22} - 9	Uni_{21} - 9				
9.	EP_1 - 1	*Uni_22* - 6	Uni_{21} - 9				

Discussion. In the presented case study for the BTr development, three versions of policy-based planning technique were used. It can be concluded that in all three versions the same scenario for the CEP construction was chosen, the same alternative EP intervals for the BTr construction were considered and all required policies were correctly evaluated. So all 8 correct BTrs were found by each version. The difference between the original policy-based planning and the ordered descending policy evaluation is in the task network decompositions structures, since the decomposition methods used in the original version were divided into domain refinements and new decomposition methods. The domain refinement methods introduce extra steps into the planning, during which policies can be evaluated for higher-level domains. These extra steps bring performance gains, when policy requests are denied at domain levels. This can be seen at the considered fragments of the decompositions structures. The same deny decision received from the university policy leads to one extra step during the descending policy evaluation and 12 extra steps during the original policy-based planning. In Table 9.3, it is shown that the number of steps required to find all plans has reduced from 674 to 469 when the descending policy evaluation is applied[10]. Another feature of the descending policy evaluation, which leads to a reduction of the

[10]Technical decompositions, which are executed along with the problem-specific decompositions in order to realise the descending policy evaluation, were not counted here. These decompositions are plain and do not involve choice

steps number, is the fact that all policy requests become fully specified when an EP interval is chosen for a compound task. Hence, if a policy request is denied at this level, it is not required to decompose such task to primitive actions and execute them. For example, this happens when for some '..._*degree*' task it is discovered that an EP cannot be studied by the student, because of the language policies. On the other hand, new domain refinement operations increase the number of policy request evaluations, since policies should be evaluated separately for different domains and re-evaluated several times if the *IndTemp* decision is received for a partially known policy request. Despite of this increase, the main characteristic, CPU time, is reduced, when the ordered descending policy evaluation is applied. The reason for this is the fact that the average time for evaluation of one policy request in the original version is $0,1s$[11]. All policies applicable to an EP should be checked during this period. Such significant amount of time is also caused by time-consuming operations with XML-based XACML policies and requests. For the descending policy evaluation, the average time of a policy request evaluation is $0,034s$, because only policies for the current domain are involved[12]. Generally, during the descending policy evaluation, when a partial policy request is evaluated at some domain, policies of this domain are evaluated for all EP intervals belonging to this domain. Only when some required information about an EP interval is missing, these policies should be re-evaluated separately for each EP interval that will be considered. On the other hand, in the original version, domain policies are always evaluated for each EP interval separately, even if during their evaluation information about the EP interval is not required. As we can see, in this case study the cumulative effect of the optimised policy evaluations and the number of steps reduction resulted in 50% CPU time reduction for the ordered version of descending policy evaluation.

Table 9.3: Three versions of the planning algorithm comparison (case study 1)

	All plans				First plan		
Version	Plans	CPU time	Steps	Requests	CPU time	Steps	Requests
ORG	8	54	674	404	8,68	25	15
DPE_{ORD}	8	26,74	469	794	7,09	22	34
DPE_{UNORD}	8	21,43	211	438	7,88	22	48

When the unordered version of the descending policy evaluation is applied, using the branching factor minimisation principle the structure of the search-space tree should be optimised, meaning that the number of nodes should be reduced. Despite the fact that we have not seen this effect in the fragment developed for the first plan (22 steps were used by both versions), when all possible plans are developed by the planner, the number of states is reduced by 55 % (see Table 9.3)[13].

points, like problem-specific decompositions.

[11] When the average time for the evaluation of one policy request is estimated, only requests that have some applicable policies are taken into account, that is, requests that produce decisions distinct from 'N/A'.

[12] Irrelevant policies are filtered at the initial step of policy evaluation using the policy targets mechanism.

[13] Moreover, we have carried out an additional experiment where 2 versions of the descending policy evaluation were compared for the problem when all policy requests are evaluated to *Permit*. The size of the full search-space

Another positive impact of the unordered planning is the possibility to determine legitimate paths on the universities level, without the need to decompose tasks to the EP level. For example, the transfer between the second and third slots is illegal for the current student if the student transfers within one university, Uni_{21} (i.e., the transfer is from one EP to another). As we have seen, this was detected in step 5 when the unordered version is used. So Uni_{21} is not considered further as a possible university in the second slot and the university Uni_{22} is chosen for this. Moreover, when during the further planning the university Uni_{12} is considered for the usage in the first slot, it is immediately rejected during the re-evaluation of the partial policy request for the $\&transfer_IT$ action in the second slot, since the transfer to Uni_{22} is not a partner transfer. Whereas, in order to reach the same outcomes using the ordered version, it is required to try all possible combinations of EP intervals within Uni_{12} for the first slot and within Uni_{21} for the second slot (6 EP intervals and one EP interval, respectively). All together, more than 78 steps were required for this[14]. The number of policy requests reduction is less than the reduction for the number of steps and is equal to 45%. Indeed, when the unordered version is used, extra policy evaluations should be carried out. After a task is decomposed, in addition to the evaluation of requests for this task, policy requests that have been evaluated to $IndTemp$ and attached to other tasks should be re-evaluated. Finally, taking into account the fact that the average amount of time required to evaluate one policy request has slightly increased to $0,035$ sec., the overall CPU time was successfully reduced from $26,7$ to $21,4$ sec.

BTrs generated by the planner in this case study should be used for the initial analysis of CEPs that can be created. Using them, a user can reformulate the planning task or select a BTr that will be used as the basis for the detailed CEPs elaboration. Hence, the user can direct the planner to a preferable area of the search space. The procedure of the detailed CEP construction based on the chosen BTr is similar to the problem considered in the next case study, where the planning task is specified using lower level domains and EPs. As was shown, the descending policy evaluation performance gains depend on policy decisions, received during the planning, and other peculiarities of the planning domain. Impacts of different factors on the descending policy evaluation performance will be evaluated using series of experiments in Section 9.4.

9.2.2 Case study 2

The second case study considers a small scale planning problem where it is required to develop a CEP with one permanent transfer. The ITr for this problem consists of one lower level domain and an EP, so this significantly limits the planner's search space for the BTr development. In this

tree, which was generated by the planner in this experiment, is reduced from 2057 to 1740 states using the unordered version. This shows that the positive effect of the branching factor minimisation principle in this case study occurs, even when the parts of the search-space are not eliminated by Deny decisions.

[14] We have counted only steps which were used to produce a fully valid transfer from $Country_1$ to Uni_{21} and have not considered steps leading to intermediate dead-ends, occurred during this search.

case study, we consider that a detailed specification of the CEP should be developed (including the process and structure models). Additionally, this case study illustrates the possibility to solve problems in a real educational environment, so most of the problem data (policies and EPs) are adopted from the real world.

Planning environment. A domain hierarchy (see Figure 9.9) represents a part of a real HE environment. A small-scale problem is considered, so only the required domains and EPs are included into the domain hierarchy. It contains three countries: Russia, UK and Ukraine. In each country, one university is presented: Bauman Moscow State Technical University (BMSTU) in Russia, Kharkov Polytechnic university (KhPU) in Ukraine and University of Birmingham in UK. Four EPs are added: three MSc, which are used for the CEP construction, and one BSc, which is used for the specification of the previous education of the student. All MSc EPs are adopted from real EPs in corresponding universities. Their complete descriptions are constructed in accordance with the EP model in Chapter 7 and placed into the planner's world state. Descriptions of these EPs are presented in Appendix D and contain description of their semesters, compulsory and optional modules, including their credit values and pre-requisites.

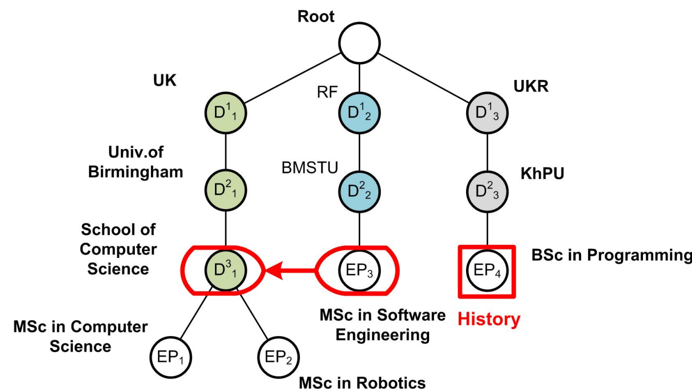

Figure 9.9: Domain tree schema (case study 2)

For the comparison of modules from different EPs in terms of their learning outcomes values, the similarity measure $sim^{mod}(Mod_i, Mod_j)$ will be used (see Chapter 7)[15] This measure is used for the recognition of modules and module's prerequisites checks during the policy evaluation. In addition to this similarity measure, in policy rules conditions the maximum similarity measure for modules $\mu^{max}_{mod}(LObj, Mod_i)$ is used to estimate the similarity of a module and a larger LObj (e.g., semester or EP interval). As was described in Section 7.2.2, this similarity measure is equal to the maximum similarity measure between the module Mod_i and some module Mod_j contained in $LObj$: $\mu^{max}_{mod}(LObj, Mod_i) = max_{Mod_j \in LObj}\{sim^{mod}(Mod_i, Mod_j)\}$. For example, in the BMSTU policy, there is a condition that a module-prerequisite is satisfied if its similarity with some module

[15]We consider values of this measure as given in this case study.

studied by the student is more than 70 %. Moreover, at a higher level of the planning, learning content-based measures are used to filter EPs that are considered for some slot based on already known EP intervals in other slots of the CEP. In concrete, when $|EP_j|$ has been chosen in the first slot, an interval of EP_i can be used in the second slot only if the value of the average between maximum similarity measures $K^{avg}_{mod}(|EP_j|, EP_i) \geq 0.5$. This measure was defined in Section 7.2.2 as an average of maximum similarity measures between $|EP_j|$ and modules in EP_i (in $|EP_j|$ only modules studied by the student are considered, in EP_i - all optional and compulsory modules)[16].

Policies. For this case study, some policies are adopted from policies in force in real domains, other policies are fictitious. As can be seen in Figure 9.10, within a policy set for a domain, policies and policy sets managing distinct aspects of the education are specified. In order to distinguish the application fields for these policies, first of all, their targets contain conditions on the action used in the policy requests and, additionally, some extra conditions can be used (e.g., on the type of resource). As a full CEP development procedure is carried out in this case study, the policies, in addition to constraints, specify obligations, which are used to construct domain-dependable task networks extensions during the planning

As example, a schema of the *Russia* policy set is represented in Figure 9.11. Its admission policy set PS_2 is composed of a policy for students with BSc degrees from Russia, Ukraine or Kazakhstan P_3 and policy for other students P_4. Rules for these policies were extracted from Russian ministry orders and Federal laws ([112, 113]): students can be admitted to an MSc degree if they have a BSc degree in the same area from a state university in Russia, Ukraine or Kazakhstan; if they have BSc degrees in the same area from other countries, these degrees should pass the official recognition procedure. Other students cannot be admitted to MSc programmes. Other policies in Russia policy set are the diploma award policy P_1 and the study policy P_2. The diploma award policy specifies the number of credits that a student should have in order for the university can grant him (or her) an MSc degree. Credits requirements are specified in credit units adopted in Russian Federation (RF) domain (one credit is equal to 36 notional hours). In order to convert credit values in credit units of other domains, corresponding transformation rules are used[17]. The study policy specifies requirements to the structure of EPs in RF: each module should be more than two credits, each semester should not contain more than 35 credits and not less than 1/8 of them should be credits earned using optional modules. These policies were extracted from State Educational Standard for MSc degrees in Software Engineering [135][18].

Other policy sets for domains are described in Appendix D. For instance, they include the

[16]When a CEP is constructed using $|EP_j|$ and EP_i, some modules in EP_i should be recognised based on $|EP_j|$ modules. The introduced requirement is aimed to filter EPs which are not similar to each other and which, therefore, cannot be used for the CEP construction.

[17]In this case study, credit units adopted in UK domain are also used. We assume that one such credit unit corresponds to 10 notional hours.

[18]Regulations specified in this standard were re-formulated in order for they can be enforced on the introduced LObj data model.

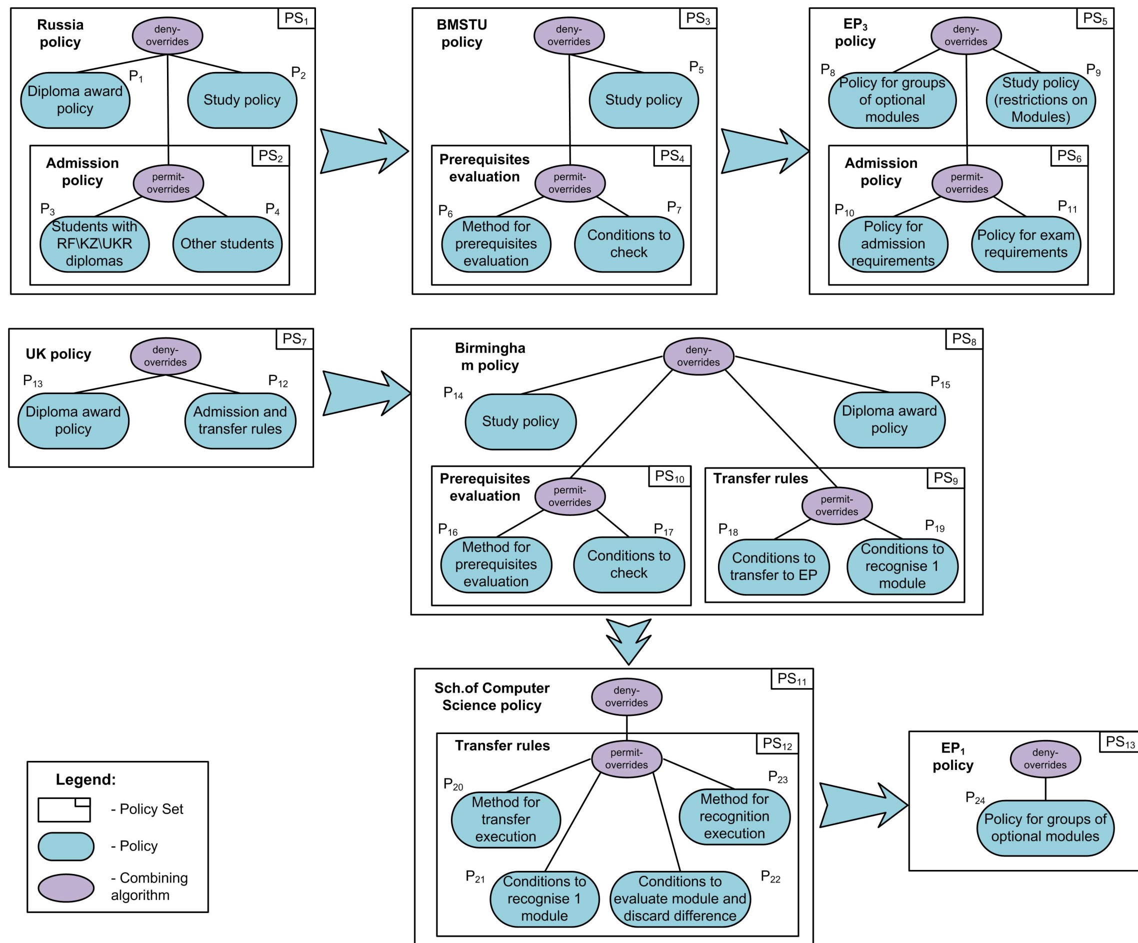

Figure 9.10: High-level policy schema (case study 2)

following policies. Prerequisites evaluation policies determine the method used to take a decision if prerequisites for an EP interval are satisfied and the condition which is checked for an individual prerequisite of a module. For example, the BMSTU policy set PS_4 specifies that for each module 60% of prerequisites must be satisfied[19]. Policies for optional modules groups are usually specified within EP policy sets. They define possible combinations of modules that should be selected in order to close this group of optional modules. The transfer policies specify the procedure that should be carried out to fulfil the student transfer, for example, if a credits recognition should be done, which type of recognition can be used (one-to-one, one-to-many or many-to-many), if the student must or can pass an additional assessment procedure in order to recognise his (or her) credits. Additionally, these policies determine conditions that should be satisfied in order to recognise a module or carry out a student transfer, for example, how many credits can/must be recognised, how many modules can be recognised by an additional assessment.

[19]The BMSTU policy for individual prerequisite evaluation was specified earlier in this section.

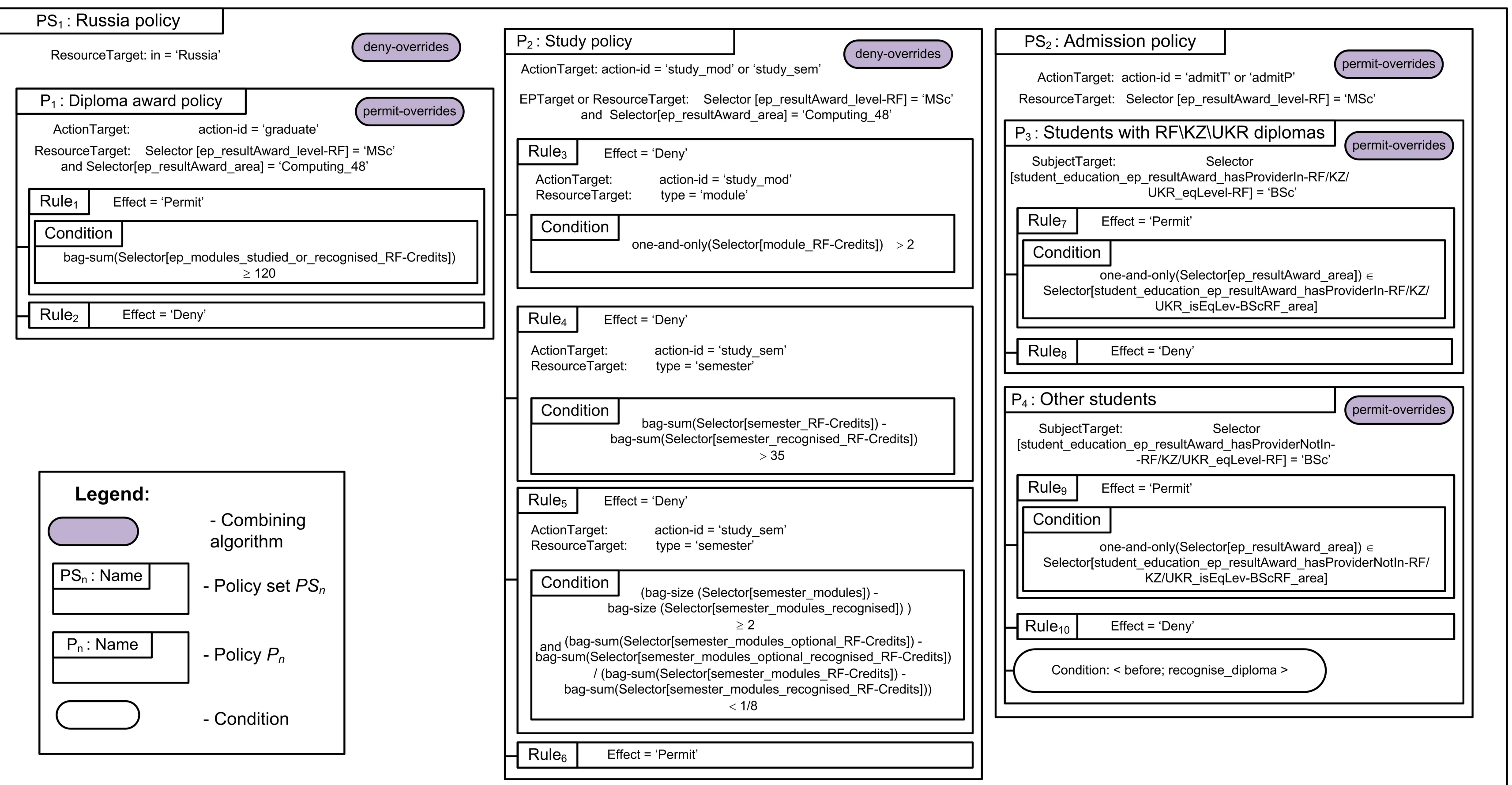

Figure 9.11: Schema for *Russia* policy set (case study 2)

Problem statement: student *Pavel* from Ukraine with a BSc in Programming from KhPU is going to enter BMSTU and study an MSc in Software engineering. However, he wants to study in two different universities during his MSc course. After some period of study in BMSTU, he means to transfer to the School of Computer science in University of Birmingham and graduate with an MSc in Computing there. The initial task, which should be solved by the planner in this case study, is $Degree(Pavel, Track_2, Award^r)$. $Track_2 = \langle EP_3, D_1^3 \rangle$ is provided as ITr. It is a 2-slots track, where in the first slot an interval of EP_3 should be used and in the second slot an EP interval can be buil from any EP within the D_1^3 domain (see Figure 9.9). The previous education and language skills of the student are specified within the initial planner's world state. The award requirement $Award^r$ specifies that the goal of the student is a UK degree in Computing (according to the ISCED taxonomy) with an MSc level (according to the UK NQF, i.e., level 7). Moreover, similarly to the previous case study, within the initial state time constraints for the result CEP are specified ($t_{Beg} = \langle 20, 08, 2011 \rangle$, $t_{End} = \langle 30, 10, 2012 \rangle$, $\delta^t = 3$).

Course of planning. As a two-slot track is used in the problem statement, the initial task *Degree* can be decomposed only by the method implementing a 'permanent transfer' scenario (see method 7.15). In turn, tasks *Start_degree* and *Finish_degree* are one-slot tasks and can be decomposed only into task networks containing primitive and compound actions. According to the ITr constraints, during the decomposition of the *Degree* task EP intervals $|EP_3|_{(1,1)}$, $|EP_3|_{(1,2)}$ and $|EP_3|_{(1,3)}$ can be chosen for the *Start_degree* task. During the decomposition of the *Finish_degree* task, $|EP_1|_{(2,3)}$, $|EP_1|_{(3,3)}$ and $|EP_2|_{(2,3)}$, $|EP_2|_{(3,3)}$ are considered, as EP_1 and EP_2 satisfy the domain ITr constraints and their awards satisfy the requirements for the target award. These EP intervals are sequentially tried during the planning. However, as we have the time limitation such that a CEP duration should be less than one year and two months, EP intervals $|EP_3|_{(1,2)}$ and $|EP_3|_{(1,3)}$ are eliminated. So $|EP_3|_{(1,1)}$ is used in the first slot. When EP_2 intervals are tried during the planning for the second slot, they are filtered by the constraint on the average of maximum similarity measures ($K_{mod}^{avg}(|EP_3|_{(1,1)}, EP_2) = 0.47$). This average value for EP_1 is equal to 0.69, so at the second slot only EP intervals $|EP_1|_{(2,3)}$ and $|EP_1|_{(3,3)}$ are considered[20]. A detailed description of the planning procedure, where these EP intervals are tried, is described further.

First, the task $Start_degree(Pavel, Track_2(1,1), |EP_3|_{(1,1)})$ is decomposed using the first method in Figure 9.12[21]. The student is admitted, modules are chosen and he studies the EP interval $|EP_3|_{(1,1)}$. Eventually, the student transfers permanently to another EP. When the first action,

[20]According to the time constraints specified, both these EP intervals can be used in the second slot, when $|EP_3|_{(1,1)}$ is used in the first slot.

[21]In the task network decomposition schema, only auxiliary actions !*concur_start*, !*concur_end* and !*change_line* are shown, which are used to define pseudo-parallel actions (see Chapter 7). Auxiliary actions *CA_start* and *CA_end* (see Chapter 5), which are used to define compound actions, are eliminated in order to simplify the schema and make it more readable.

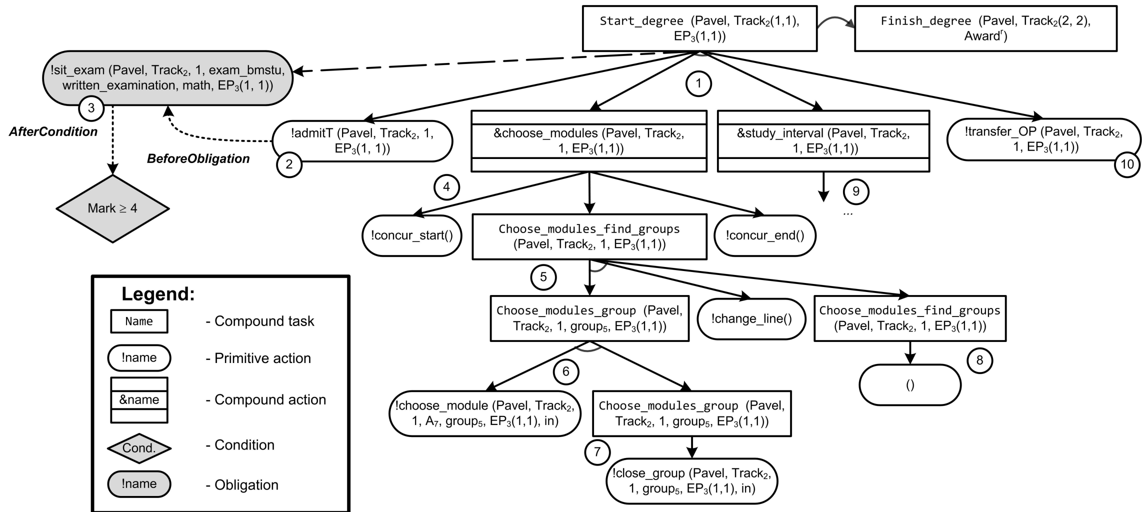

Figure 9.12: Task network decompositions structure in case study 2

!*admitT*, is executed (point 2 in Figure 9.12), the policy request is permitted by the corresponding policies within EP_3 and *Russia* policy sets (P_{10} and P_3). Additionally, the admission policy for EP_3 specifies that students must pass an exam in math. So the before-obligation !*sit_exam* is generated during the policy evaluation. A new policy request for the !*sit_exam* action is created during its execution. According to the policy P_{11}, this request is evaluated to Permit and an after-condition is generated designating that the student's mark for this exam should not be less than 4 (point 3 in Figure 9.12). When the before-obligation !*sit_exam* has been executed, preconditions and the policy request for !*admitT* are re-evaluated. It is checked that their outcomes are not changed during the before-obligation execution.

Next, the compound action &*choose_modules* is decomposed and executed (see point 4 in Figure 9.12). This compound action is decomposed into the task *Choose_modules_find_groups*, which searches for groups of optional modules within the current EP interval and processes them in separate threads using pseudo-parallel actions. Since $|EP_3|_{(1,1)}$ contains only one group of optional modules, this task contains in the decomposition structure only one thread. For each group found, a task *choose_modules_group* is generated, which recursively tries to select different combinations of modules, until the group can be closed. A policy request for action !*choose_module* is evaluated to Permit if this module can be chosen, considering already chosen modules from this group. A policy request for the action !*close_group* is evaluated to Permit if the current group can be closed according to the policy. For the optional modules group in $|EP_3|_{(1,1)}$, containing modules A_7 and A_{11}, the policy P_8 specifies the following rules: only one module can be chosen from the group and the group can be closed if one (and only one) module is chosen. During the execution of the *choose_modules_group* task (see point 6), first, a module A_7 is chosen and the policy request for the !*close_group* action is permitted (see point 7), so the group is closed and

other its modules cannot be chosen. The planning goes further, but when the !*close_group* and !*choose_module* actions and the corresponding decompositions are backtracked, module A_{11} is tried. The compound action &*study_interval* models a study period, carried out according to the EP interval $|EP_3|_{(1,1)}|$. It is decomposed into one compound action $\&study_sem(\dots, |EP_3|_{(1,1)}|, \dots)$, representing the education of the student in one semester: the first semester of EP_3. In turn, this task is decomposed into a pseudo-parallel actions, at each thread of which the student studies one module (it is modelled using an action !*study_mod*). The details of the process of &*study_interval* action execution is described in Section 9.3.2 (the schema of the &*study_interval* decomposition is shown in Figure 9.15). During its execution, it is checked that according to the policies the student can study all modules. Additionally, during the evaluation of policy requests for actions !*study_mod*, before-obligations are generated initiating the module prerequisites evaluation. The prerequisites evaluation method and specific conditions that should be checked are specified in the policy set PS_4 using obligations and have been described earlier. During the execution of these obligations, it is confirmed that prerequisites of obligatory and selected modules in $|EP_3|(1,1)$ are satisfied.

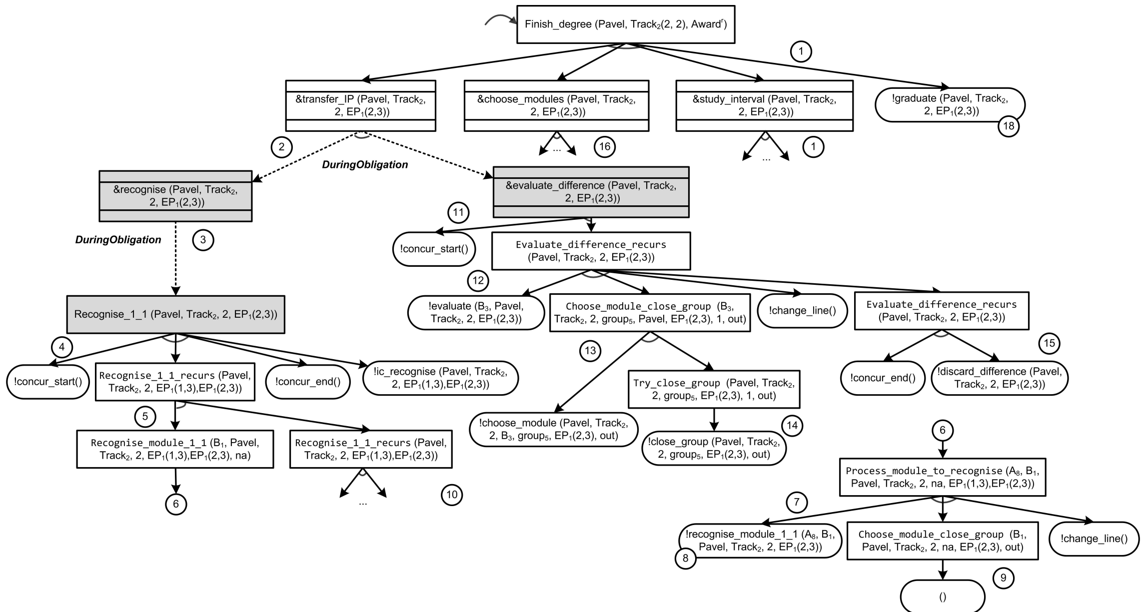

Figure 9.13: Task network decompositions structure in case study 2 (cont.)

The primitive action $!transfer_OP$ is used to designate a permanent transfer, meaning that the student has transferred from BMSTU and will not return to it in the future. There are no policies for this action, so it is evaluated to N/A. The $Finish_degree$ task is decomposed into an ordered task network (see the first decomposition in Figure 9.13), consisting of $\&transfer_IP$ compound action, which models the process of transferring the student to University of Birmingham with the aim to graduate from it, and compound actions &*choose_modules* and &*study_interval*, applied

to the EP interval $|EP_1|_{(2,3)}$. At the end of this task network, action $!graduate$ designates the graduation of the student from EP_1 and granting him the corresponding award.

Transfer rules are defined in the University of Birmingham by faculties. The policy P_{20} of the Computer science school always permits compound action $\&transfer_IP$, but generates two during-obligations, hereby defining the routine according to which this transfer should be carried out (see point 2 in Figure 9.13). The compound action $\&recognise$ indicates that modules of EP_1 can be recognised during the transfer (if the student has studied equivalent modules in another EP). After the possible recognitions are carried out, the next compound action $\&evaluate_difference$ is used to estimate the difference that could remain between the interval $|EP_1|_{(1,1)}$, which the student should have studied, and the recognised part of $|EP_1|$. This task also provides the possibility to pass extra exams in order to decrease this difference and make the transfer possible. The policy P_{23} specifies the procedure for the recognition. According to it, when a policy request with the action $\&recognise$ is evaluated, the during obligation *Recognise_1_1* is returned. This task defines the recognition routine during which only one-to-one modules recognition can be carried out. The task *Recognise_1_1* produces pseudo-parallel actions processing one module in one thread. Recursively, it processes all modules in the EP interval $|EP_1|_{(1,3)}$[22] using tasks *Recognise_1_1_recurs* and *Recognise_module_1_1*. The former task implements the recursion. The latter task is applied to the module which is recognised in the current thread. During its execution, all modules that the student has studied are tried as possible variants for the recognition. So, the next task $Process_module_to_recognise$ is applied to two modules: the module that can be recognised and the module that was studied and can support this recognition (see point 6). The latter modules, which were used to support a recognition, cannot be used for further recognitions. Therefore, the method for the decomposition of *Recognise_module_1_1* goes over all possible studied modules and stops when a new variant of recognition is found. If the new variant of recognition is not found, the planning goes forward, as not all modules must be recognised according to the task *Recognise_1_1*. In this manner, all possible variants of the recognition can be tried one by one. When an optional module is recognised, additionally, the action $!choose_module$ is executed and a trial to close the corresponding modules group is carried out[23]. In our case study, the policy P_{21} permits to execute an action $!recognise_module_1_1(Mod_1, Mod_2, \dots)$ when $sim^{mod}(Mod_1, Mod_2) \geq 0.7$ and the number of credits for Mod_1 is equal or greater than Mod_2's credits. The first possible variant of recognition is $A_8 \rightarrow B_1, A_3 \rightarrow B_2, A_1 \rightarrow B_{11}, A_7 \rightarrow B_4, A_4 \rightarrow B_5$. During the execution of the

[22] During the transfer, modules from both intervals of the EP, which the student transfers to, can be recognised: the interval of the EP that the student is going to study and the EP interval that the student should have studied before the transfer according to the EP.

[23] In Figure 9.13, point 7, a variant when the compulsory module B_1 is recognised is represented. In tasks *Recognise_module_1_1*, *Process_module_to_recognise*, *Choose_module_close_group*, which are carried out for its recognition, dummy *na* parameter is used instead of the optional modules group. Therefore, the task *Choose_module_close_group*, which is designed to select an optional recognised module and try to close the group where it is contained, is decomposed into an empty task network.

task &$evaluate_difference$, using the action !$discard_difference$ it is checked that the minimum requirement for the recognised modules are satisfied and the transfer can be carried out. In case not enough modules have been recognised, during the execution of &$evaluate_difference$ action the student can pass additional assessments based on which more modules can be recognised (this is designated by the action !$evaluate$). The policy P_{22}, which defines constraints for the execution of !$discard_difference$ action, requires that all modules that the student should have studied according to the EP interval $|EP_1|_{(1,1)}$ (i.e., a part of the EP before the transfer) must be recognised. The only module that is not recognised within this EP interval is B_3. However, this module contains only 10 UK credits, so, according to the policy P_{22}, the student can pass an exam and this module will be recognised for him. Therefore, during the execution of the method decomposing the &$evaluate_difference$ compound action, the action !$evaluate$ is executed for this module (see point 12)[24]. As a result, all required modules are recognised for the student and the action !$discard_difference$ is permitted for the execution by the policy P_{22} (see point 14). Next, when the compound actions &$choose_modules$ and &$study_interval$ are executed, the action !$graduate$ is evaluated according to the policies P_{13} and P_{15}. The policy P_{13} requires that the student should have not less than 180 UK credits (or an equivalent amount of credits in other scales). The policy P_{15} requires that not less than a half of these credits is granted for modules studied by the student in the University of Birmingham. These two conditions are satisfied, so the action !$graduate$ is permitted and the award of EP_1 is granted to the student.

Table 9.4: CEP structure generated (case study 2)

Semester	University	*Module* / Credits								Sum
1	BMSTU	A_1	A_2	A_3	A_4	A_5	A_6	A_7 (Opt.)	A_8	RF Cr.
		4	4	4	3	4	3	5	3	30
2	Univ. of Birmingham	B_6 (Opt.)	B_7 (Opt.)	B_8						UK Cr.
		20	10	30						60
3	Univ. of Birmingham	B_9								UK Cr.
		60								60

The process model of the designed CEP is presented in Figure 9.14. The corresponding CEP structure is presented in Table 9.4. During backtracking, for the same BTr, $\langle |EP_3|_{(1,1)}, |EP_1|_{(2,3)} \rangle$, another possibility for the recognition of modules is found. In $|EP_1|_{(1,1)}$, modules B_4 and B_{11} can both be recognised using the module A_1 as well as A_7. Therefore, in the new recognition variant, modules that are used for the recognition of B_4 and B_{11} are swapped. The process model for the second variant is updated correspondingly, but the CEP structure model is not changed. During the further backtracking, $|EP_1|_{(3,3)}$ is tried in the second slot. So semesters 1 and 2 of EP_1 should be recognised during the transfer. However, according to the recognition rules in the policy set

[24] After a module is evaluated, it is processed using the same procedure, as when it is recognised. When the module is optional (e.g., module B_3), it is chosen and a trial to close the corresponding modules group is carried out.

PS_{12}, this cannot be done because the difference in modules content is too large. When this is detected, the backtracking is continued and, finally, when it comes to the action &*choose_module* within the *Start_degree* task, the module A_{11} is chosen from the group of optional modules in $|EP_3|_{(1,1)}$, instead of A_7. With the updated set of modules in $|EP_3|_{(1,1)}$, the planning goes further, but during the execution of the &*study_interval* compound action it is detected that the module A_{11} cannot be studied by this student[25]. Thereby, the found CEP with the two possible process models is the only CEP, which can be created to solve the problem in this case study.

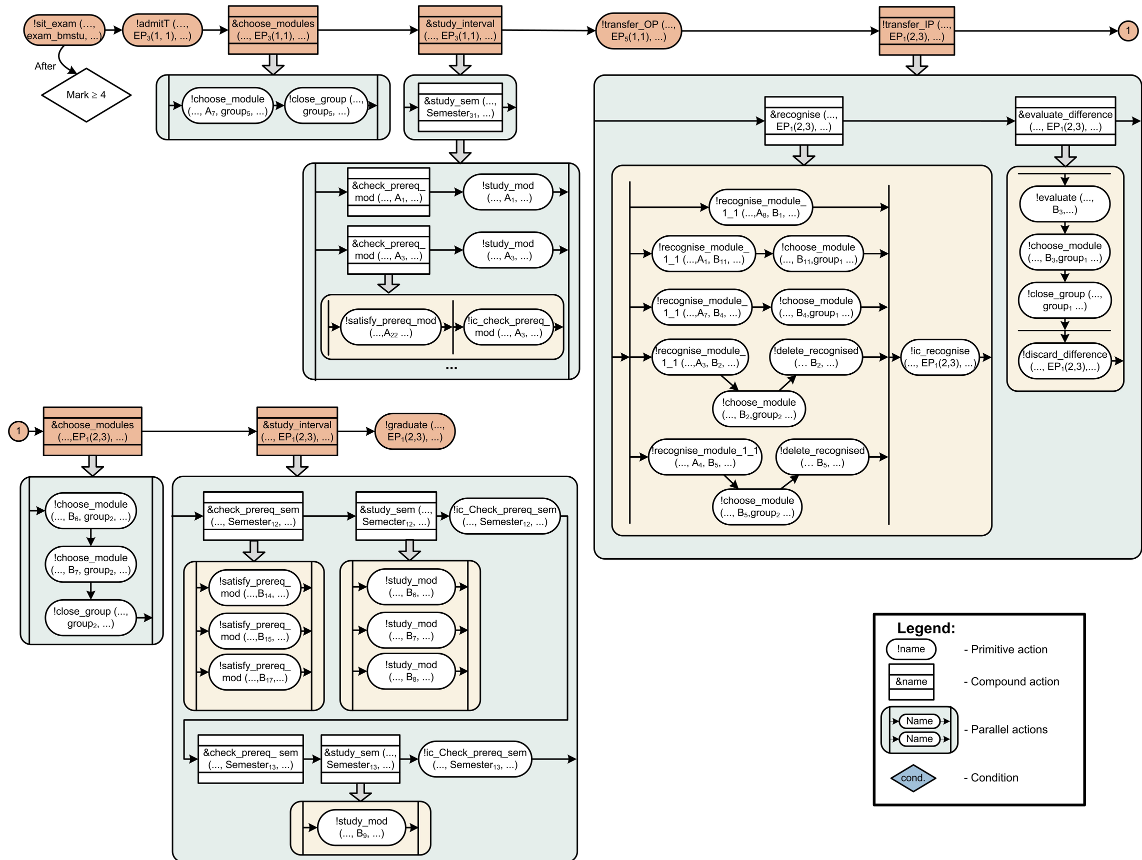

Figure 9.14: CEP process model generated (case study 2)

9.3 Planning in environments with heterogeneous regulations

The policy-based planner was designed to provide the means to use the automated planning techniques in environments where regulations, which determine legitimate processes, are composite, applicable only in certain contexts and are specified by different persons. The policy-based planner provides this possibility using the policy language as a tool for the specification of such regulations. As was demonstrated in the case studies, the XACML policy language can be used to specify educational regulations that possess the described properties of the heterogeneity. These regula-

[25] According to the policy P_9, the student studying this module should be a citizen of RF.

tions were specified as separate policies and policy sets, which manage different aspects of the educational process and are applicable only in certain contexts. These policies can be specified by different persons independently, because during the policy evaluation policy decisions, returned by different policies, are combined using policy combination algorithms, which determine the routine for the conflict resolution. Moreover, different policies and policy sets can be specified in different files. In this section, we will analyse the possibilities of the policy-based planner to enforce relevant policies depending on the current situation during the planning.

9.3.1 Enforcement of relevant policy constraints

In this section, we analyse how different sets of constraints can be specified and enforced during the planning and possible levels of granularity of these constraints. For this purpose, we use the planning environment specified for the first case study. In that case study, policies were used to specify different constraints on the educational processes modelled using the planning environment. As was shown in Section 9.2.1, the same policies are evaluated and the same plans are generated by all three versions of the policy-based planning algorithm, so in this section we consider only the original policy-based planning algorithm.

Using this planning environment, we observe which factors can result in changes of the policies being enforced during the policy evaluation. For this purpose, 9 scenarios with different planning problems were created. In these scenarios, different ITrs and students, for which a CEP should be developed, are used. In addition to $Student_1$, used in the first case study, $Student_2$ and $Student_3$ are introduced. These students have certificates from $Country_2$ and aim to get a degree in the area of Computer Science issued in $Country_2$ with a level equivalent to the first cycle of the EHEA QF (BSc). $Student_2$ knows languages $Lang_2$ and $Lang_3$, while $Student_3$ knows only $Lang_2$. We have considered in detail the evaluation of two policy requests and collected policy decisions returned during the evaluation of these policy requests in different scenarios runs. Some of these policy decisions along with the policy requests are presented in Table 9.5. Using information about these decisions, we can see which policies are enforced during the policy evaluation at specific points during the planning (i.e., which policies have returned non-*N/A* decision).

First of all, a policy is enforced only if the policy request contains a relevant EP: an EP that is a descendant for the domain of this policy. As can be seen in Table 9.5, admission policies for Uni_{11}, Uni_{12} and Uni_{13} return non-*N/A* decisions only for policy requests (with the !*admitT* action) where an EP from the corresponding university is used[26]. The action, which is used in the policy request, also determines the policies that should be enforced. When an action in a policy request matches an action in the policy target, this policy becomes applicable. For example, admission policies and study policy sets return non-*N/A* decisions only when the corresponding

[26]When *N/A** decision is used, the fact that the policy is irrelevant was detected using higher-level policies, e.g., an *N/A* decision was returned by a target of the whole university policy set.

Table 9.5: Policy decisions received during execution of different scenarios

Scenario			Policy request		Policy decisions					
#	Student	ITr	EP at 1 slot	Action	Pol. Admit Uni_{11}	Pol. Admit Uni_{12}	Pol. Admit Uni_{13}	PS Study $Country_1$ Target	PS Study $Country_1$ P home student	PS Study $Country_1$ P int. student
1	$Student_1$	$\langle Uni_{11}, Country_2, Uni_{21} \rangle$	EP_1 or EP_2 or EP_3	admitT	*Permit*	*N/A**	*N/A**	*N/A*	-	-
				study_int.	*N/A*	*N/A**	*N/A**	Applic.	*Permit*	*N/A*
2	$Student_2$			admitT	*Deny*	*N/A**	*N/A**	*N/A*	-	-
3	$Student_3$			admitT	*Deny*	*N/A**	*N/A**	*N/A*	-	-
4	$Student_1$	$\langle Uni_{12}, Country_2, Uni_{21} \rangle$	EP_4 or EP_5 or EP_6	admitT	*N/A**	*Permit*	*N/A**	*N/A*	-	-
				study_int.	*N/A**	*N/A*	*N/A**	Applic.	*Permit*	*N/A*
5	$Student_2$			admitT	*N/A**	*Permit*	*N/A**	*N/A*	-	-
				study_int.	*N/A**	*N/A*	*N/A**	Applic.	*N/A*	*Permit*
6	$Student_3$			admitT	*N/A**	*Permit*	*N/A**	*N/A*	-	-
				study_int.	*N/A**	*N/A*	*N/A**	Applic.	*N/A*	*Deny*
7	$Student_1$	$\langle Uni_{13}, Country_2, Uni_{21} \rangle$	EP_7 or EP_8 or EP_9	admitT	*N/A**	*N/A**	*Permit*	*N/A*	-	-
				study_int.	*N/A**	*N/A**	*N/A*	Applic.	*Permit*	*N/A*
8	$Student_2$			admitT	*N/A**	*N/A**	*Permit*	*N/A*	-	-
				study_int.	*N/A**	*N/A**	*N/A*	Applic.	*N/A*	*Permit*
9	$Student_3$			admitT	*N/A**	*N/A**	*Permit*	*N/A*	-	-
				study_int.	*N/A**	*N/A**	*N/A*	Applic.	*N/A*	*Deny*

action is used in the request. Hence, actions in policy targets designate phases of the process modelled during the planning when this policy should be enforced. Furthermore, different policies can be enforced based on finer grain aspects (e.g., different properties of the student, the EP or the action). For example, within the study policy set for $Country_1$, two policies are distinguished: for home students and international students. In the policy targets of these policies, a condition on the citizenship property of the subject is used. Hence, different policies are enforced for $Student_1$ from $Country_1$ and for $Student_2$ and $Student_3$ from $Country_2$.

In order to illustrate other cases of independent policy enforcement, we run three more scenarios in the same planning environment. These scenarios are run for $Student_2$ and different ITrs (see Table 9.6). We analyse policy decisions returned by the university partner network policies for policy requests with $!transfer_IT$ and $!transfer_IP$ actions in slots 2 and 3. As can be seen from the table, different policies are enforced depending on the EP which was studied by the student immediately before the policy request. If the previous EP and the EP where the student is going to transfer to are from the same university, the policy for inner-university transfer is enforced. Otherwise, the inter-university transfer policy is employed within the partner network policy to derive a decision about legitimacy of the student transfer. Therefore, different policies can be enforced based on the information about actions that have been executed by the planner before the policy request evaluation. This history information can be utilised if it has been saved as the effect of an action in the planner's state and has any relations with designated objects in the policy request being evaluated.

Hence, during the policy-based planning, when an action is executed, different policies can be enforced depending on the planner's action itself and all information about this action that can be included into the policy request, generated by the policy-based planning mechanism for this

Table 9.6: Policy decisions received during execution of different scenarios (cont.)

Scenario		History		Policy request		Policy decisions			
						PS Partner network Uni_{21}		PS Partner network Uni_{22}	
#	ITr	EP at 1 slot	EP at 2 slot	EP used	Action	P Inner transfer	P Inter transfer	P Inner transfer	P Inter transfer
1	$\langle Uni_{12}, Uni_{21}, Uni_{21}\rangle$	EP_6	-	EP_{10} or EP_{11}	transfer_IT (slot 2)	*N/A*	*Permit*	*N/A**	*N/A**
		EP_6	EP_{10}	EP_{11}	transfer_IP (slot 3)	*Permit*	*N/A*	*N/A**	*N/A**
2	$\langle Uni_{13}, Uni_{21}, Uni_{21}\rangle$	EP_7	-	EP_{10} or EP_{11}	transfer_IT (slot 2)	*N/A*	*Deny*	*N/A**	*N/A**
3	$\langle Uni_{13}, Uni_{22}, Uni_{21}\rangle$	EP_7	-	EP_{13}	transfer_IT (slot 2)	*N/A**	*N/A**	*N/A*	*Permit*
		EP_7	EP_13	EP_10 EP_{11}	transfer_IP (slot 3)	*N/A*	*Permit*	*N/A**	*N/A**

action. This information includes the action name, action parameters, objects used as designated objects for this action. Moreover, all information that can be retrieved about the designated objects from the planner's world state can also affect the policy applicability: their binary properties, like citizenship of the student, or relations with other objects and information that can be extracted about these objects, like education providers of EP intervals studied by the student. Information inferred using transformation rules can also be taken into account, like number of credits assigned to a module that can be converted to credit units of different countries. The history about the execution of actions, which have been carried out by the planner, can be taken into account if it is saved into the planner's world state as effects that have some relations with designated objects in the policy request. Changes in this information can result in the enforcement of different policies, as within targets elements of policies, constraints on this information can be specified limiting the policy applicability. In the considered case study, the hierarchical organisation of policies is used: first of all, all policies are divided based on the domains that they are applied to. The domains form a hierarchy and one action corresponds only to one leaf-domain, so conflicts can occur only between policies corresponding to domains laying on a path from the root domain to the leaf-domain. Conflicts between these policies are resolved in a way that Deny decision takes precedence, that is, in order to execute an action, it should be permitted (or an N/A decision can be produced) by all these policies. Within the domain policies, first of all, constraints on action names are used to distinguish different policies, next, other finer grain constraints can be used. In general, other criteria with different priorities and different number of levels can be used to define a hierarchical policy structure or other policy structures can be specified using XACML targets and policy nesting mechanisms (e.g., a flat structure, where only one policy can be applicable to a policy request). Different organisations of policies determine when a specific policy can be applicable and how conflicts are resolved, when several policies return decisions for the same policy request.

9.3.2 Enforcement of established routines

In this section, we analyse the possibility to enforce different established routines using the policy-based planner. In a planning environment with heterogeneous regulations, in order to carry out the same task, different procedures can be used in different domains or even within the same domain. For example, in the second case study, different routines can be carried out when the student transfers from one university to another. They can differ in a way how credits are recognised and how a decision about the student transfer is taken. In terms of the HTN planning, this results in a requirement to decompose the same task into different task networks, depending on the domain where it is executed and, possibly, other factors. Moreover, rules that determine the resulting task network can be specified by different persons independently.

In the policy-based planner, the task networks generation mechanism, which is based on policy obligations, is developed in order to support planning environments with such regulations. We will examine the task networks generation mechanism using the second case study, where it was used for the specification of routines for the evaluation of prerequisites. When a CEP is developed for a student, the prerequisites of modules should be checked. However, usually some number of unfulfilled prerequisites can be permissible. The number of unfulfilled prerequisites can be evaluated on the level of modules, on the level of semesters or EP intervals. Hence, different routines for the estimation of unfulfilled prerequisites and for taking the decision if a student can study this EP interval can be specified. For example, assume that in BMSTU prerequisites should be evaluated for each module separately. In the BMSTU policy set, it is specified that for the !*study_mod* action a before-obligation &*check_prereq_mod* should be generated (see Figure 9.15). This prescribes that before studying a module, a prerequisites evaluation procedure should be applied. Moreover, for the compound action &*check_prereq_mod*, different routines can be used as well. For example, in BMSTU, not all prerequisites for a module must be fulfilled, specifically, in order to study a module, only 60 % of its prerequisites should be met. In the BMSTU policy set, this is specified using a rule that for the &*check_prereq_mod* action a during-obligation $Satisfy_prereq_mod$ should be generated. The task $Satisfy_prereq_mod$ is used to enforce the described rule. During its execution, all prerequisites of the module are checked and, then, the action !*IC_check_prereq_mod* is executed. Policy constraints for this action are used to check if the required number of the module's prerequisites is satisfied. A fragment of the task network representing one semester study period at BMSTU is shown in Figure 9.15.

In contrast, the University of Birmingham prerequisites are evaluated on the level of semesters in the considered case study. In the corresponding policy set, there is a policy that for each compound action &*study_sem* a before-obligation &*check_prereq_sem* and an after-obligation !*IC_check_prereq_sem* should be generated (see Figure 9.16). During the execution of the &*check_*

Figure 9.15: Task network representing one semester study period, generated within the BMSTU domain

prereq_sem compound action, all the prerequisites of modules that should be studied in the corresponding semester are evaluated. The action *!IC_check_prereq_sem* is executed after the *&study_sem* compound action. It is used to finally decide if all conditions for studying this semester, including the prerequisites, were satisfied by the student[27]. In the University of Birmingham policy, there is a condition that this action is permitted only if the sum of credits for the unfulfilled module-prerequisites is less than 30 UK credits. As we can see, different task network structures were generated in these two domains during the execution of same task. These differences have appeared due to different obligations, specified for the involved actions in the domain policies.

Therefore, using the task networks generation mechanism of the policy-based planner, different task networks can be generated depending on obligations specified in the policies being enforced. The task networks produced during the decomposition are extended using before- and after-obligations generated during the policy evaluation for actions contained in this task network. During-obligations extend the task network in depth and form a new level of the task network decomposition. In the policy-based planner, obligations generated during the policy evaluation can form their own task network, with the corresponding ordering constraints. They are united with the task network of the planning method being applied using HTN planning constructs. Therefore, structures of task networks being generated in policy-based planning are determined by two specifications: HTN planning methods, specified in the HTN planning environment, and policy obligations, specified in policies. The HTN planning methods are applicable throughout the whole

[27]It is executed after the *&study_sem* action because during the execution of *&study_sem* a set of modules that a student will study in this semester can be changed. In some domains, modules that student cannot study according to the policies can be removed from the EP.

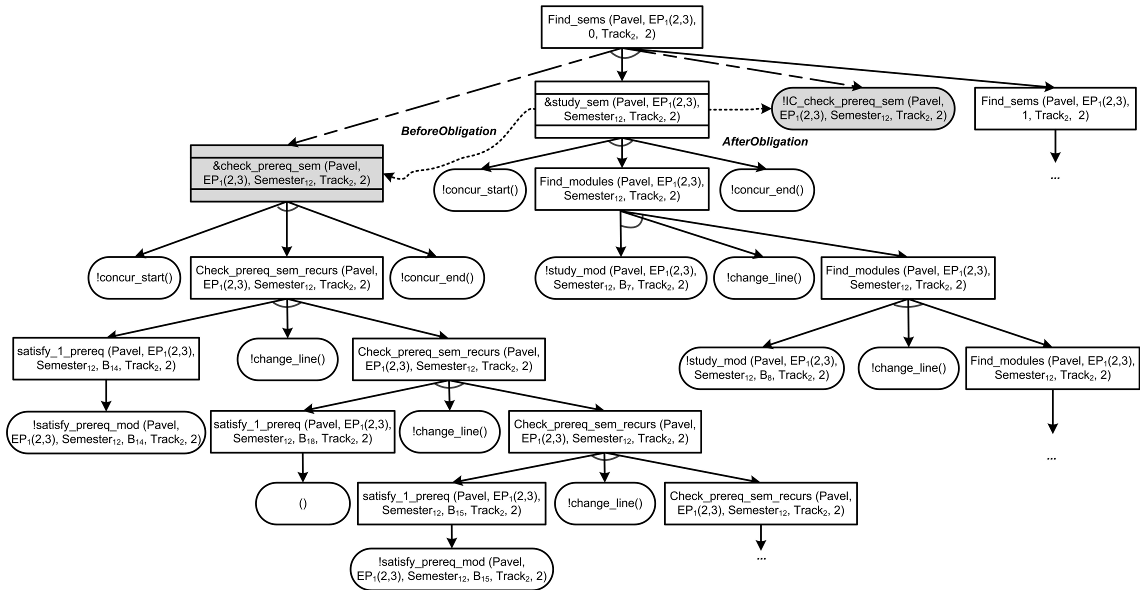

Figure 9.16: Task network representing one semester study period, generated within the University of Birmingham domain

environment and are changed only with the HTN planning environment, while obligations, specified as part of policies, possess all useful policy properties. They can be changed dynamically, while the planner is running. Obligations can be specified by different persons independently, as part of different policies. Moreover, they are enforced only in specific situations, when the whole policy, containing it, is applicable in a current situation. The possible level of granularity for obligations specification is the same as for policy constraints, which were analysed in Section 9.3.1, because an obligation is tied to the policy where it is specified. In order to specify which obligations can intervene into the planning process when a certain action is executed, the obligations validation mechanism was developed. Using this mechanism, it is possible to define the set of task networks that can be returned by the policy engine in response to a policy request with a specific action.

9.3.3 Planning in an environment with dynamically changing regulations

Heterogeneous regulations are supported by different persons independently, hence, they can be updated by them in an uncoordinated manner. Policies used in the policy-based planner can be updated even when the planner is running. First of all, these dynamic updates of policies are possible, because the whole policy set is a composition of separate policies and policies sets, which can be managed and updated independently (e.g., different files are used for the specification of policies authored by different persons). Secondly, dynamic policy updates are possible, as policies are specified in a declarative form using the policy specification language, externally from the planner and its planning domain specification. So, as opposed to the planning domain, which should be rigidly designed as a single whole and is contained as an integral part of the planner,

individual policies can be updated dynamically and asynchronously. The planner checks policy updates when it is running. If an update or deletion of some specific policy is detected, an updated policy is loaded into the system and the outdated policy is deleted. These policy updates are applied during the planning for the next planning problem. The only restriction, which was purposely imposed, is the following: when a specific problem is being solved, an unalterable policy set should be used, otherwise the solution developed can be inconsistent.

9.4 Performance analysis

In this section, the analysis of performance gains achieved by two versions of the descending policy evaluation technique, in comparison to the original policy-based planning procedure, is presented. These performance gains were already shown in the discussion part of case study 1 (see Section 9.2.1). It was shown that performance of different versions of the policy-based planning algorithm is determined by the planning problem, including policy characteristics and HTN decomposition tree characteristics. In this performance analysis, we take into account different characteristics of the planning problem and, by varying it, make conclusions about its effects on the performance (similarly to [185, 87]).

We adopt the planning problem of case study 1 as the basis for the planning problem used in this performance analysis. Several modifications were introduced to this planning problem, in order to make it more homogeneous, in order to eliminate incidental impacts on performance. The domain tree of the planning problem used in the analysis is a regular tree with branching factor 3 (it is the same as the tree in Figure E.1, A., but with 3 EPs in each university). Domains used as constraints in the ITr are all situated on the same level of the tree (countries level) and are all distinct. Therefore, the same number of EPs is considered when the planner is searching an EP interval to be used in different slots. One of the most influential factors on the policy-based planning performance are decisions being generated during the policy evaluation. In order to control this factor, we have substituted XACML policy sets specified for domains and EPs by random variables that takes a value *Deny* with probability p and *Permit* with probability $1 - p$ (therefore, this variable has Bernoulli distribution). Each time a policy request should be evaluated against a policy set of specific domain (or EP), the value of this variable is used to determine the policy evaluation outcome. Probability p is called *policy stringency*. Using these 'simulated' policies definitions, policies with different stringencies can be easily modelled using the p value. If we relate the policy stringency value with real policies, it corresponds, for example, to the ratio of the number of possible values defined in a policy for some attribute to the number of specific objects used in the planning domain that can be used as a value of this attribute. In concrete, in the partner network policy in case study 1, policy stringency corresponds to the ratio of the number of universities, from which a transfer is denied, to the overall number of universities in the domain:

$p_{transfer} = \frac{|\mathbf{UNI}/\mathbf{P}_{Tr}^{Uni}|}{|\mathbf{UNI}|}$, where **UNI** is set of all universities, $\mathbf{P}_{Tr}^{Uni}$ - the set of universities from which a transfer is permitted according to the policy. The evaluation of real XACML policies is associated with computational costs, so the prototype's computations are suspended for an amount of time required for a policy decision inference. In our experiments, we use delay values equal to the average time intervals for a policy request evaluation found in case study 1, considering the version of the planning algorithm.

Another policy characteristic affecting the performance is the ratio of *IndTemp* decisions and permanent decisions, being returned for partial policy requests during the evaluation. We also model this policy characteristic using random variables, that have Bernoulli distribution, with two outcomes: if an *IndTemp* decision is returned during the evaluation or not. The probability to receive an *IndTemp* decision as a result of the policy evaluation is determined by the amount of information that is absent in the partial policy request. Therefore, we fix the probability to get an *IndTemp* decision for a partial policy request with specific properties as a reference value: it is a request where all information is known excepting the EP interval used as a resource. The policy characteristic that determines the probability to get an *IndTemp* decision for such partial policy request is designated z and will be called reference *IndTemp* probability. Based on $\alpha = 1 - z$, we will determine the probability τ to get a permanent decision for all policy requests using formula $\tau = \frac{\alpha}{k} + \sum\limits_{i=1,(k-1)} \frac{\alpha}{k} \cdot Q_i + (1-\alpha) \cdot Q_k$, where k is the number of the slot for which policy request is evaluated and Q_i - variable indicating if an EP interval is known in slot i (it is equal to 1 when it is known, 0 otherwise). This formula was designed based on the following premises, when a policy request is fully known τ should be equal to 1, meaning that *IndTemp* cannot be produced during the evaluation of this request. When a partial policy request contains all information excepting the EP interval used as a resource, it should be equal to α. For each unknown EP interval before slot k, the τ value should be reduced by a constant amount (i.e., $\frac{\alpha}{k}$). When no EP intervals are known, τ should not be equal to 0, since policies can utilise other information contained in the policy request, excepting the information about EP intervals (in our case, this minimum τ value is $\frac{\alpha}{k}$). Varying z, we can model how policies process partial policy requests. *IndTemp* decisions are generated with the probability $1 - \tau$. Therefore, the probability to a get *Deny* decision is equal to $\tau \cdot p$.

9.4.1 Policies characteristics impact analysis

In the first series of experiments, we analyse how policy characteristics influence the policy-based planning performance, the CPU time required to find a solution of a planning problem. Using the policy evaluation simulation mechanism, we can model policies with different properties (i.e., p and z values). During the run-time, *Permit* or *Deny* decision is chosen as an outcome of the policy request evaluation using a generated random number from 0 to 1, whose value is compared

with the current p value. As we use the same planning tasks for all runs, these random numbers fully determine a set of resulting plans, found by the planner. In order to compare performance of different versions of the planning algorithm, they should solve exactly the same planning problems. We achieve this by running our planning problem with the same *seed* that is used to generate random numbers for the choice between *Permit* and *Deny* decisions during the evaluation of a specific policy request (the same decisions will be generated for the same policy requests when different versions of the planning algorithm are used to solve the same planning problem).

Experiment 1.1. Policy stringency p changes. In this experiment, we analyse how the mean CPU time changes for planning problems with different values of policy stringency p. The z value is fixed in this experiment and is equal to 0. The utilisation of the same p and z values guarantees that policy decisions appear with the same probabilities, but specific policy decisions received during the planning (especially for domains) have big impact on the planning process. In order to analyse the mean planning performance, we have selected 10 random planning problems (they are specified as 10 different seeds for the *Permit* and *Deny* generation), for which at least one plan can be found when the highest p value concerned is used[28]. We run these 10 problems using the three versions of the planning algorithm with different p and measure the CPU time, the number of policy requests evaluations and the number of planner's states traversed during the planning (i.e., the number of recursive calls of the planning function). Two latter characteristics are used for explanation purposes for the main characteristic, the CPU time. The average values for these 10 problems were taken for the analysis. Graphs of these characteristics are presented in Figure 9.17[29]. As performance gains can be different for planning problems where it is required to find all possible plans and problems when planning stops after the required number of plans is found, two values for each characteristic were measured: the final value when all possible plans are found and the value when 16 plans were found[30]. As can be seen from the figure, absolute values are decreasing with the growth of p for the all-plans planning problems, since for the greater p less number of plans can be found (the average number of plans is shown in rectangles on the plot). However, for planning problems with a limited number of plans, values are increasing since with the growth of p each plan becomes more difficult to find.

As can be seen in the graphs for CPU time, the descending policy evaluation technique brings considerable performance gains in comparison with the original policy-based planning technique.

[28]The maximum p value is 0.15, since for larger values of the policy stringency very few planning problems have at least one solution.

[29]As the CPU time is influenced by many factors and can vary for the same runs, each planning problem is run 3 times with the same settings and the average time for these 3 runs was taken.

[30] The number of plans was limited in order to analyse the behaviour of the planner when it successfully finds the required number of plans in comparison with the unrestricted case, when the required number of plans is not limited. Moreover, in order contrast this scenario with the unrestricted case and prevent the traversal of the whole planner's search space, the plans limit value should indent from the maximum number of possible plans. Therefore, the value 16 was chosen in a way that for all p values the planner should find the required number of plans and for the maximum p value the plans limit should be approximately the half of the overall number of plans.

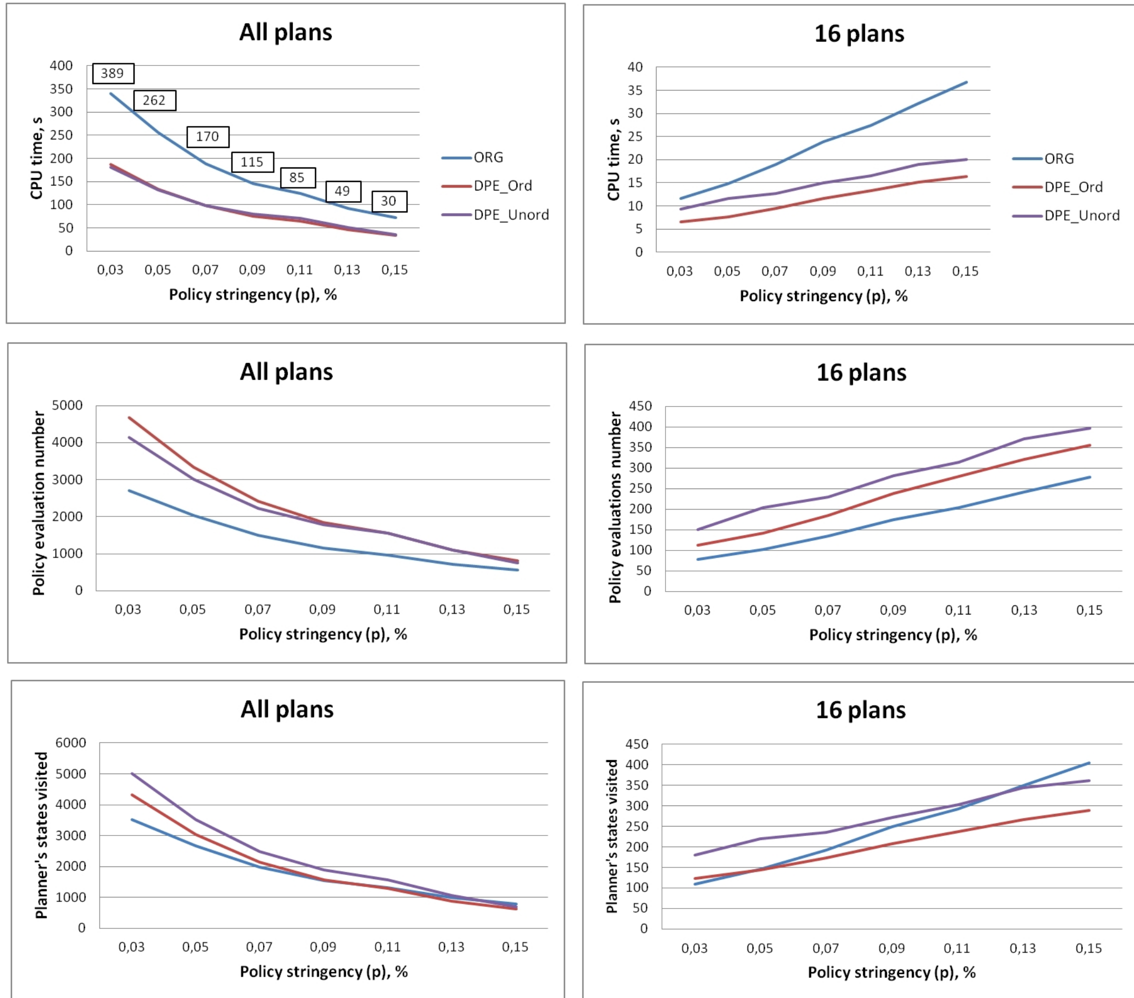

Figure 9.17: Experimental results for different values of policy stringency p

Ratios of CPU time values for the ordered descending policy evaluation to values of the original policy-based planning varies from 0.45 to 0.55 (see Figure 9.18) and the difference between full or limited problems is not big. The unordered version of the descending policy evaluation has CPU time values similar with the ordered version for the full problems for all p. Meanwhile, it has higher CPU time values in comparison with the ordered version for problems with the limited number of plans. Nether-the-less, the CPU time for this version never exceeds the CPU time for the original policy-based planning algorithm. As can be seen, time ratios for both versions of the descending policy evaluation for the full and limited problems have a downward trend when p increases, since when *Deny* decisions occur more frequently the descending policy evaluation is more advantageous. Therefore, both versions of the descending policy evaluation require less CPU time to solve the same planning problems than the original version of the policy-based planner. Moreover, the descending policy evaluation is more beneficial when the policies are more strict.

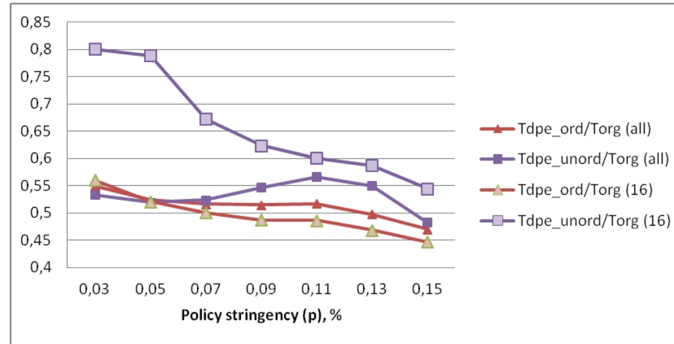

Figure 9.18: CPU time ratios for different values of policy stringency p

In order to understand the reasons of such CPU time ratios, the number of planner's states visited and the number of policy requests evaluated should be analysed. The number of planner's states for all-plans problems for the ordered descending policy evaluation is approximately equal to the corresponding values when the original planner version is used, for the most of p values, but exceed these values for p less than 0.07. This results from the fact that in general the descending policy evaluation requires extra planner's states, but when *Deny* decisions occur they are detected earlier in the state-space (because domain and even EP policies are evaluated for compound tasks). This means that some states, which should be generated in order to detect these dead-ends when the original policy-based planning is used, are not produced. This forms the gain of the descending policy evaluation technique. The unordered descending policy evaluation increases the number of states for all p values. This happens because for the current planning problem the shape of the decomposition AND/OR graph, which determines the shape and size of the planner's search space, is not optimal for the FAF strategy. The similar fact was examined in [158], where it was shown that on average the FAF heuristics decreases the number of states during the planning, but there are cases when it fails to do this. This happens when the FAF principle prescribes to postpone a refinement and each option of this refinement starts a long chain of refinements without branches. In our problem, such refinements are decompositions of the '$\ldots_degree$' compound tasks into several actions, which should be sequentially executed by operators (the application of an operator does not introduce options, so the branching factor is equal to one). During the ordered planning, these tasks are decomposed immediately when they are produced, on the higher levels of the planner's state-space tree. However, when the unordered planning and the FAF strategy are used, as the number of possible options for such decompositions[31] is usually greater than the number of possible decompositions for other tasks, these decompositions are postponed for later. However, before these decompositions are executed, decompositions with branching factors more than one can be carried out. Therefore, the number of times these decompositions are carried out during

[31]It is equal to the number of possible EP intervals.

the planning is multiplied by the branching factors of all decompositions that have been executed before them.

When it is required to find the first 16 plans, the number of states ratios for both versions of the descending policy evaluation to the original policy-based planning algorithm are better than the corresponding ratios for the all-plans problems. When p is greater then 0.05, the number of states for the ordered descending policy evaluation is even less than the corresponding value for the original planning algorithm (the same is true for the unordered version with p more than 0.11). The reason of this behaviour is the fact that when the planner searches for an EP in some slot and the number of plans is limited, the planner stops at some suitable EP and does not try all possible EPs (often this can be the first EP that satisfies the corresponding requirements). This leads to the change in the ratio of *Deny* and *Permit* decisions occurring during the planning: more *Deny* decisions occur and, therefore, more EPs are rejected by the planner, relatively to the overall number of EPs considered. As we have shown before, the increase in the number of *Deny* decisions is beneficial for the descending policy evaluation performance (this is valid for *Deny* decisions generated at the domain level, as well for *Deny* decisions generated at the EP level, since additional decompositions of compound tasks to action tasks can be avoided then). Similarly with the experiment when the number of plans is unlimited, the unordered descending policy evaluation shows worse results in the number of states traversed than the ordered descending policy evaluation. However, when only the first 16 plans are considered, the results of the unordered versions are even worse. This happens because the minimisation of a state-space graph that the FAF strategy is trying to achieve can give profits only when the whole state-space graph is explored.

Graphs for the number of policy requests replicate graphs for the number of states with different interrelations. Ratios between the ordered descending policy evaluation and the original version increase from $0.8-1.2$ up to $1.2-1.5$, because most of the states in the descending policy evaluation correspond to several policy evaluations, while in the original policy-based planning only some states require a policy evaluation. The ratios of the unordered descending policy evaluation to the ordered version are decreased. Such results were shown because states that produce 'long chains', which have prevented the unordered version from the reduction of the state-space size, do not require any policy evaluations at all. This means that during the planning, in spite of not-reduced state-space graphs, the number of policy evaluations decreases by the utilisation of the FAF strategy. In fact, for the full planning problem, the unordered version of the descending policy evaluation generates equal or less number of policy requests than the ordered one. Differences between the CPU time graphs and policy evaluation graphs are caused by the different amounts of time required for a policy request evaluation in different versions of the planning algorithm. The relative position for two descending policy evaluation graphs are kept similar, while the original version, due to large time expenses on policy evaluations, significantly exceeds the other versions

in CPU time.

Experiment 1.2. Indeterminate Temporal decision rate z changes. In this experiment, we vary the reference $IndTemp$ probability z, which determines the number of policy requests for which a permanent policy decision can be generated in presence of incomplete information (based on information about the current domain, the student and his (or her) history of education[32]. Therefore, it indicates the amount of information that is required in order to produce a permanent decision during the policy evaluation. The experiment schema was the same as in the previous experiment. The same 10 seeds were used for the generation of random values for the $Permit$ and $Deny$ decisions determination, and additional 10 seeds were used in these problems for the generation of random values for the $IndTemp$ decision generation. These problems were run on 3 versions of the planning algorithm with different z values. Since during the original policy-based planning only fully known policy requests are used, the z value has no effect on it. Thus, this version was run only once for each problem. Furthermore, we have carried out this experiment for 4 different values of p to analyse how z influences the performance in the presence of policies with different stringency. The resulting diagrams are presented in Figure 9.19[33]. In the figure, we show only graphs for all-plans problems since the same trends exist in graphs for 16 plans problems. In order to analyse the changes of the observed values, for each value with $z > 0$ the ratio to the corresponding value with $z = 0$ is calculated. We analysed how these ratios change with the growth of z. For this purpose, we have divided these ratios received using the same version of the descending policy evaluation algorithm into sets with fixed p and applied the regression analysis to each of this set. We used the linear regression model where the dependent variable is the ratio and the explanatory variable is the z value. Values of regression coefficients for z, determined using the ordinary least squares method for these sets, are shown in Figure 9.19 above the column charts.

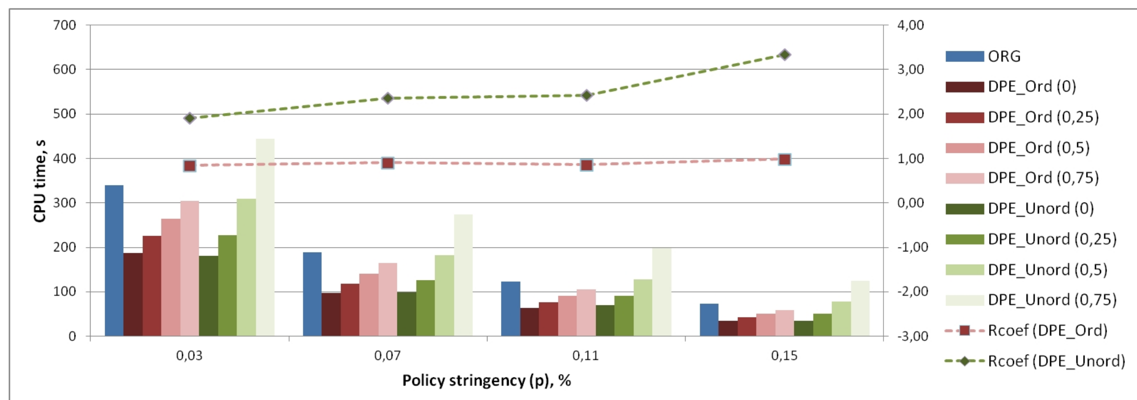

Figure 9.19: Experimental results for different values of z (see also Figure E.2)

[32]In some policy requests, the information about the history of education can also be unavailable.

[33]Since $IndTemp$ decisions are resolved into $Deny$ or $Permit$ decisions which are under the control of p value, the number of plans generated in each planning problem is determined by p and is not influenced by z (the average number of plans for these p values can be found in Figure 9.17)

As is shown in Figure E.2 in Appendix E, the number of planner's states increases with the growth of z. When an *IndTemp* decision is generated due to the absence of some information, utilised in policies for the evaluation of a policy request, the planner continues planning and traverses the next states. However, later, when more information is available, this policy request can be resolved to *Deny*. In this case, all states visited by the planner from the *IndTemp* decision occurrence up to the *Deny* disclosure constitute planner's overheads. As can be seen from the column and the regression coefficient graphs, the number of states increases with the growth of z faster for the unordered descending policy evaluation. This happens because during the ordered descending policy evaluation the planner at one time considers only one EP interval selection problem, for one slot. Therefore, immediately after the evaluation of policies for an EP interval, the planner chooses the EP interval itself. So all *IndTemp* decisions occurred during the evaluation of domain policies are resolved into permanent decisions during the next 1 - 2 states. When the unordered version is used, after an *IndTemp* decision occurred, the planner can choose to switch to another slot and this *IndTemp* remains unresolved for longer. Thus, the average lifetime for *IndTemp* decisions during the unordered policy evaluation is longer. Moreover, during the unordered planning, the probability of *IndTemp* decision is higher since in addition to current EP interval previous EP intervals can also be unknown during the policy evaluation. The regression coefficient for both versions of the descending policy evaluation grows with the increase of p. This happens because overhead states occur when *IndTemp* is resolved into *Deny* and this situation occurs more frequently when policy stringency is higher.

The same patterns can be detected in the number of policy evaluations diagram (see Figure E.2), with the difference that the values are growing faster. This tendency occurs because each partial policy request that was evaluated as *IndTemp* is re-evaluated in further planner's states, where new information for its evaluation is added, until the *IndTemp* decision is resolved. Since the number of policy evaluations has greater impact on the CPU time, the growth of the CPU time with the increase of z repeats the growth of the policy evaluations number (see Figure 9.19). In spite of the negative effects of the *IndTemp* probability increase, the ordered descending policy evaluation shows better CPU time results than the original policy-based planning for all considered z values. The unordered version of the descending policy evaluation shows better CPU time only for small p and z values: for values greater than $p = 0.03$ and $z = 0.25$ it shows worse results than the ordered version. Moreover, for $z > 0.5$, it requires more CPU time even than the original policy-based planning algorithm (for $z = 0.5$ this happens only for $p \geq 0.11$).

9.4.2 Domain tree characteristics impact analysis

In this section, it is analyse how characteristics of the domain tree influence the performance of the policy-based planner. It should be noted that modifying the domain tree characteristics, we

also change the planning problem that should be solved. For example, changing the number of EPs, we change the number of plans that can be found by the planner.

Experiment 2.1. Number of EPs changes. In this experiment, we analyse the performance of the planner operating with different number of EPs. We change the number of EPs in each lower-level domain from 3 EPs, which were used in the previous experiment, to 6 and 9 EPs. The resulting planning problems are solved using different versions of the planning algorithm, with different p values. The z value in this experiment is fixed and is equal to 0.25. As in the previous experiments, in order to estimate the average performance we use 10 different planning problems, specified as 10 pairs of seed values (one seed for the *Permit* and *Deny* decisions generation and another - for the *IndTemp* generation). Based on the values provided using these 10 planning problems, average values are calculated. They are presented as column diagram in Figure 9.20.

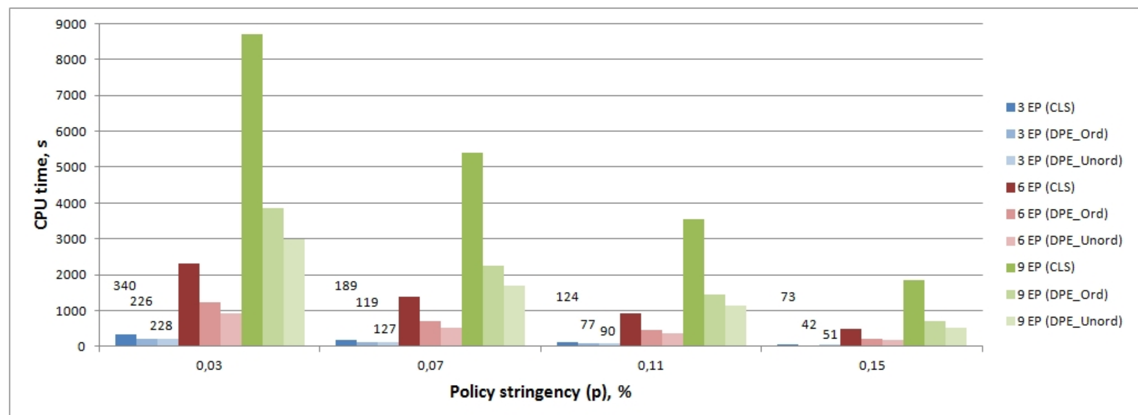

Figure 9.20: Experimental results for different number of EPs (see also Figure E.3)

Obviously, when the number of EPs is increased, more plans can be found in planning problems with the same p value. Therefore, all values demonstrate a growth in the diagrams. Furthermore, as can be seen in Figure 9.20 and from the graphs for the CPU time ratios in Figure 9.21, when more EPs are available, the CPU time ratios for both versions of the descending policy evaluation are improved. For 3 EPs, the ratios are in the interval $0.57 - 0.74$, for 6 EPs $0.53 - 0.35$ and for 9 EPs $0.28 - 0.44$. Such results are shown because, when each domain contains more EPs, if a *Deny* decision is received during the evaluation of policies for the domain, more EPs are eliminated. As usual, with the growth of policy stringency, the time ratios for both versions of the descending policy evaluation are decreasing. Importantly, with the growth of the EPs number, the unordered descending policy evaluation starts to outperform the ordered version. For the number of EPs greater than three, the unordered descending policy evaluation for all three versions and all p values shows the best results.

In order to reveal the reasons of such behaviour, we need to refer to diagrams for the number of states and policy evaluations (see Figure E.3). The unordered descending policy evaluation

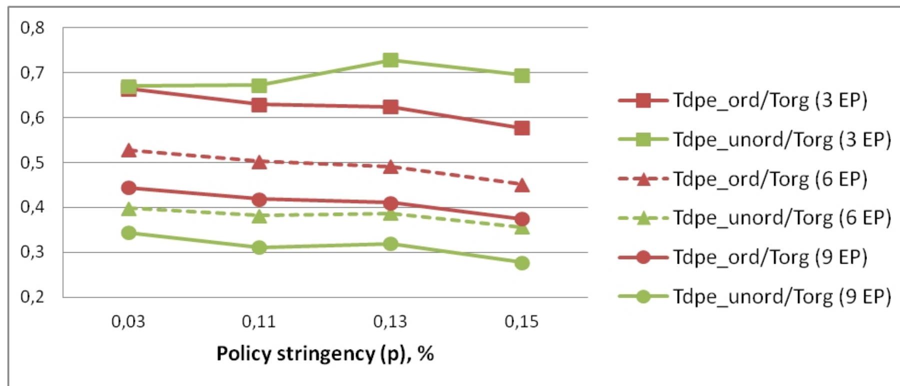

Figure 9.21: CPU time ratios for different number of EPs

traverses less states for 6 and 9 EPs. This happens because, when the number of EPs is greater, decompositions that produce long chains of states without branches and that precluded the unordered version from the number of states reduction, have greater branching factors ($K_b r$). These decompositions are decompositions of '... _degree' tasks into actions, so their branching factors are equal to the number of EP intervals from one university that can be used in current slot, according to the ITr constraints. Since the difference with branching factors for other tasks increases, the FAF strategy, which tries to minimise the branching factors, becomes more advantageous in comparison with the ordered version, which applies the decompositions with large $K_b r$ earlier. The mapping of the number of states diagram to the policy evaluations diagram is similar to the corresponding mappings in the previous experiments. The number of policy evaluations for the original policy-based planning becomes markedly less than the number of evaluations for the ordered descending policy evaluation. The ratio of the unordered descending policy evaluation to the ordered one is decreased, because, as it was said before, states within long non-branched chains, which prevent to reduce the number of states in unordered planning, do not contain policy evaluations. Finally, the CPU time diagrams in Figure 9.20 resemble the diagrams for the policy evaluations with the following difference. Regardless the number of EPs and the p value, the original policy-based planning demonstrates worse results than other planning algorithms.

Experiment 2.2. Domains tree structure changes. In this experiment, we analyse the planner's performance for problems with domain trees that have different structures. We vary the branching factor K_{br} and the number of levels N_{Lev} in the domain tree. As the basis, we have taken the planning problem with 6 EPs in each university from the previous experiment. First, the impacts of the branching factor changes are analysed. For this purpose, we have varied the number of universities in each country[34]. Planning problems with K_{br} equal to 1, 2, 3, 9 and 18 are built. The number of EPs in each country was kept constant in order to keep the complexity of the

[34]The number of universities in our problems is equal to the branching factor value.

planning problem constant as well. Thus, for the problem with $K_{br} = 1$, all 18 EPs are situated in one university. With the growth of K_{br}, the number of EPs in one university is decreasing. For $K_{br} = 18$, each of 18 universities has only one EP. Examples of the domain trees with $K_{br} = 3$ and 2 are presented in Figure E.1, A. and B. In the next phase, for the planning problem with $K_{br} = 2$, we have modified the number of levels in the tree. First, we have added the level of departments (see Figure E.1, C.). Next, above the universities level, we have added a level of regions. During these modifications, K_{br} was kept constant, so the number of domains at the lower level of the domain tree gradually increased (while keeping the overall number of EPs constant). Similarly as in the previous experiments, 10 random problems were used to get the average values.

When we are increasing the K_{br} value, the CPU time for the descending policy evaluation is increasing, relatively to the values for the original policy-based planning (see Figure 9.22). This happens because with the growth of K_{br}, extra domains are added to the domains tree. This forms extra overheads, since they should be traversed during the descending policy evaluation. On the other hand, the probability to reject an EP based on the domain policies is kept constant and is equal to p, so benefits from the introduction of these new domains are absent. Moreover, if a *Deny* decision is produced at a domain level, the smaller number of EPs is eliminated. Hence, the results of experiment have shown that the descending policy evaluation produces better results for problems with domain trees with smaller K_{br} value. However, in this experiment, by keeping the p value constant, we have assumed that the same number of policies is allocated in each university domain, regardless of the current number of domains. In reality, if we would like to reduce the number of domains in a domain tree, we will need to add extra policies to the remaining domains. For example, we are using a domain tree with the structure '*Country* $\rightarrow$ *Department* $\rightarrow$ *EP*'. If we would like to substitute departments with universities, we will need to allocate the department policies to the university nodes, in addition to the university policies allocated there already.

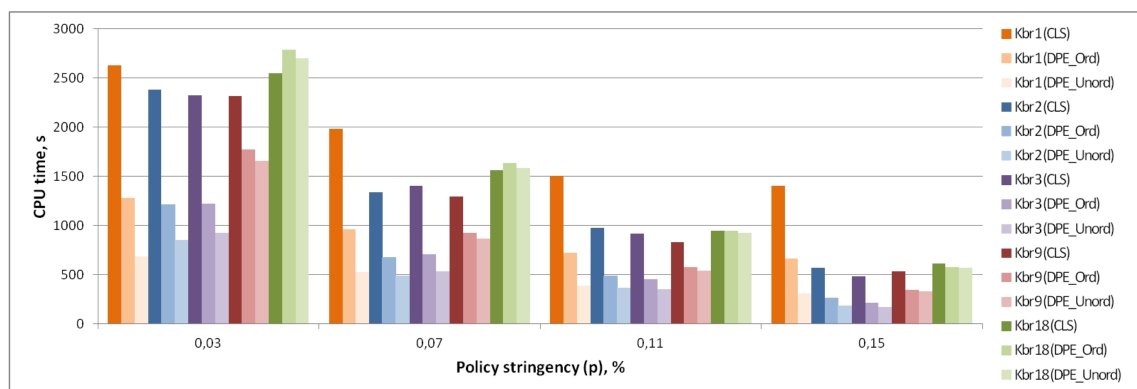

Figure 9.22: Experimental results for different values of K_{br} (task with 2 levels)

In the next part of this experiment, we modify the number of levels in the domain tree, adding

one or two new levels. Therefore, we assume that new policies should be specified at the new levels. In order to correctly simulate the time required for the evaluation of policies, we need to adjust the delay time modelling the policy decision inference (see Section 9.4.1)[35]. In order to get the average CPU time required for a policy request evaluation for problems with 4 and 5 levels, we have added the required number of levels to case study 1, have specified policies for these new domains (equal to the university policies) and carried out the required measurements. The average CPU time that we have got for the original policy-based planning have increased up to 0.13s for 4 levels and to 0.16s for 5 levels respectively. As could be predicted, the average CPU time for a policy request evaluation have not changed significantly for both versions of the descending policy evaluation. These values are used as policy evaluation delays during the current experiment.

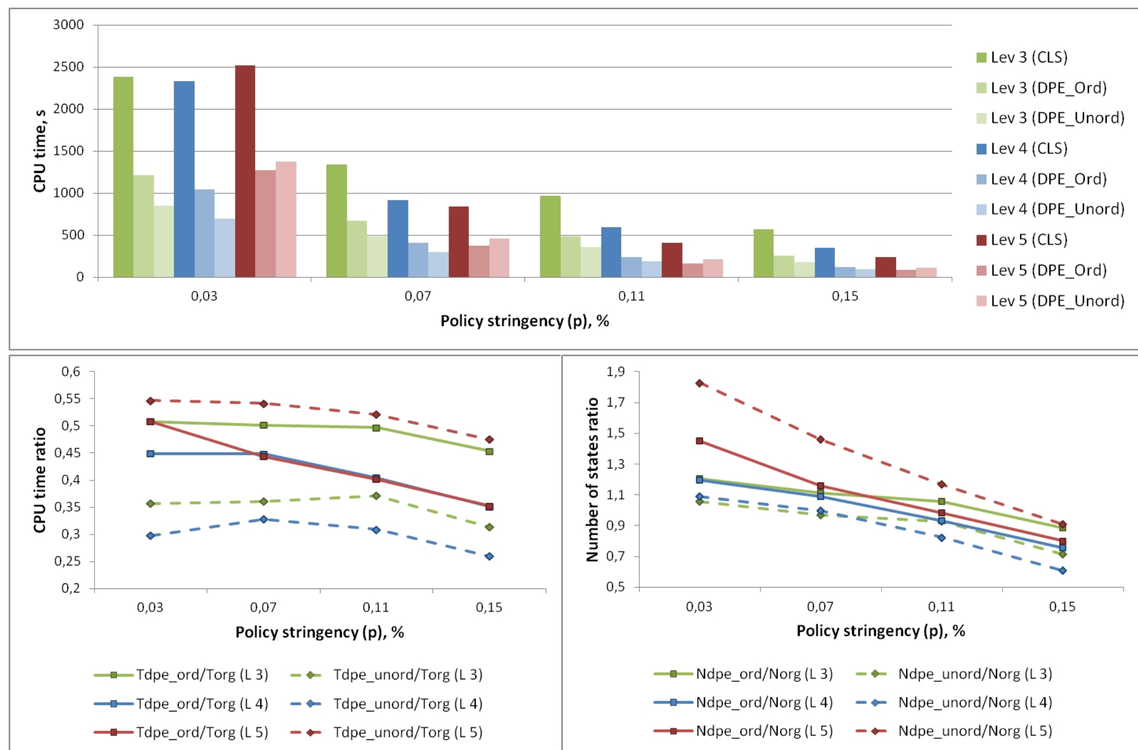

Figure 9.23: Experimental results for number of levels N_{Lev} ($K_{br} = 2$)

The CPU time diagrams for planning problems with different numbers of levels N_{Lev}, as well as ratios graphs are presented in Figure 9.23. With the introduction of new levels, new domains are added also into the domain tree. This leads to the increase of the number of states, because for each domain a distinct state should be used during the descending policy evaluation. On the over hand, because, along with these domains, new policies are added into the policy repository, the probability that an EP will be rejected because some policy at the domain level is evaluated as *Deny* increases.

[35] Delay time values that were used in the previous experiments were measured for case study 1, which has 3 levels in the domain tree.

This tends to reduce the number of states during the descending policy evaluation with large p values. Therefore, as is shown using the number of states ratio graphs, for the ordered descending policy evaluation, with the growth of N_{Lev} the ratio for small values of p is approximately the same or getting greater. While for larger values of p, this ratio is decreased or kept around its previous value. For the unordered descending policy evaluation, the same tendency exists, but, in addition, with the growth of N_{Lev} number of states ratio is kept approximately the same or getting worse. This happens because with the introduction of extra levels, we decrease the number of EPs in the lowest domains. This reduces the difference of K_{br} values for lower and higher domains and makes the domain tree more homogeneous. Thus, the advantages of the unordered planning with the FAF strategy become weaker. As a result, for $N_{Lev} = 5$, the unordered descending policy evaluation requires more states than its ordered version.

The changes of graphs for the number of policy evaluations in relation to the graphs for the number of states are the same, as in the previous experiments: the ratio between the ordered descending policy evaluation and the original policy-based planning is increasing and the ratio between the unordered descending policy evaluation and the ordered one is decreasing. CPU time values in addition to the planner's states and policy evaluations numbers are influenced by the average time required for one policy request evaluation. This value increases for the original policy-based planning with the growth of N_{Lev}, so this influences positively on the presented ratios for CPU time. As a result, with the growth of N_{Lev}, the changes of the CPU time are similar to the changes of the ratio for the number of states (i.e., the negative trend observed when the number of policy evaluations is considered is levelled). Moreover, as opposed to the number of states ratios, when the unordered descending policy evaluation is used, CPU time for problems with 4 levels is better for all p than CPU time values for the ordered version[36]. However, for problems with 5 levels in the domain tree, the unordered version shows the worst results for all N_{Lev}. The reason of this is the large number of states generated by the planner for these runs due to the homogeneous structure of the domain tree. Therefore, it can be concluded that addition of new levels with corresponding new policies improves the ratios for the ordered descending policy evaluation. These improvements are explained by an increase in probability for rejection of an EP based on its domain policies and by an increase of time for one policy request evaluation during the original policy-based planning. However, as with the growth of N_{Lev} number of domains grows exponentially and the number of EPs in each domain decreases, these improvements are appreciated only up to some limited number of levels in a tree. Moreover, the described changes of the domains tree shape result in weakening of the advantages for the unordered descending policy evaluation, because the shape of the tree becomes more homogeneous.

[36]The reason of this change is the fact described in Section 9.4.1: policy requests are not generated for the long state chains without branching that have prevented the unordered version of the descending policy evaluation to decrease the number of states

9.5 Conclusion

The main contribution of this chapter is the fact that the feasibility of the CEP generation solution for the development of CEPs based on existing EPs in environments with heterogeneous regulations was confirmed. For this purpose, first of all, we utilise case studies where CEP development problems are solved using the designed CEP generation solution. The first case study uses a planning environment where only fictitious EPs and policies are used. In this case study, we considered the problem of BTr development, for a mobility scenario with two transfers. Additionally, in this case study three different planning algorithms were tried and compared: the original policy-based planning technique, the ordered descending policy evaluation and the unordered version of the descending policy evaluation. Initial comparative analysis of these three algorithms illustrated performance gains achieved by the descending policy evaluation techniques and showed the necessity for a more detailed analysis of their performance in planning environments with different characteristics. The second case study is a small scale case study, which considers a student mobility scenario with one transfer. For this case study, a considerable part of the planning environment was adopted from the real educational environment (including domains, EPs and policies). In this case study, fully specified CEPs, represented by the process and structural CEP models, were developed.

Additionally, we have analysed the properties of the CEP generation solution that were specified as input requirements for its development, that is, the possibility to carry out planning in environments with heterogeneous regulations. Using specially developed scenarios, we have analysed the possibilities for independent specification and enforcement of policies during the planning, provided by the means of the policy-based planning mechanism designed. We have shown which information from the planner's state can be utilised to decide if a policy is applicable to a specific planner's state and, correspondingly, if the decision of this policy should be enforced during the policy-based planning. Additionally, we have analysed the possibility to enforce established routines, which are specified using policies, during the planning. For this purpose, the task networks generation mechanism based on the policy obligations was explored. Finally, the possibility to carry out planning in environments with dynamically changing regulations was analysed.

The additional contributions of this chapter are the results of the performance analysis carried out for different versions of the policy-based planning algorithm. In the experiments conducted, we have simulated the XACML policy decision generation using random variables with the known distributions. Using this mechanism, we were able to analyse how different policy characteristics impact on the planner's performance. First of all, it was revealed that the descending policy evaluation technique can decrease the CPU time required to solve a planning problem, in comparison with the original policy-based planning algorithm. The main reasons for this outcome are the

reduction of time required for one policy request evaluation and the possibility to reject all EPs within one domain, when during the evaluation of policies for this domain a *Deny* decision is produced. Consequently, as was shown in all experiments, performance gains of the descending policy evaluation technique are increasing with the growth of the policy stringency. The exceptions where the descending policy evaluation failed to improve the CPU time were problems with large domain tree branching factors. It was revealed that the growth of the branching factor has a negative impact on the descending policy evaluation's performance. In problems where the branching factor K_{br} was so large that in each lowest domain only one EP was allocated, both versions of the descending policy evaluation technique required more CPU time than the original policy-based planning. It was also shown that when the probability to get an $IndTemp$ decision increases, the CPU time required to solve the problem using the descending policy evaluation techniques is increasing. Additionally, it was found out that this negative impact is stronger for the unordered version of the descending policy evaluation.

The ratio between CPU time for the ordered descending policy evaluation and the unordered version varies. In addition to the $IntTemp$ probability, it depends on the shape of the domain tree, where the planning problem is being solved. The most influential positive factor for the unordered descending policy evaluation performance is the number of EPs, allocated in the lowest domains. When this value increases, the time ratio between the unordered descending policy evaluation and its ordered version decreases and the unordered version starts to show the best results among all three versions of the planning algorithm. Additionally, it was found out that the homogeneity of the domain tree has a negative impact of the performance of the unordered descending policy evaluation. When a domain tree (which includes EPs as well) has a more homogeneous structure, meaning that equal branching factors are used for all nodes in the domain tree (and, therefore, in the corresponding task decomposition structure), the unordered descending policy evaluation shows worse results.

Chapter 10

Conclusion

Objectives:

- *Summarise the thesis.*
- *Revisit the original contributions.*
- *Revisit the success criteria.*
- *Give details about possible future work.*

10.1 Summary

Student mobility is a rapidly developing area, being promoted at both national and international levels. While its development is beneficial for students and universities, nowadays there is no technical solution providing support for the student mobility programmes development process.

In this thesis, a novel Combined Educational Programmes (CEP) generation technique for the student mobility support was developed using both automated planning technology and policy-based management. This technique provides possibilities to plan for new CEPs in an environment with heterogeneous educational regulations, which govern different aspects of the educational process and are specified by different authors independently (e.g., education providers). Our investigation of current e-Learning technologies, specifically planning-based Curriculum Generation (CG) techniques, revealed that development of curricula in such environments is still an unresolved issue. In CG techniques, as well as in automated planning, it is assumed that the environment specification is elaborated by one author or a group of authors in a close collaboration.

Based on these findings and based on the undertaken analysis of the student mobility area, an overall CEP generation framework was constructed and, using it, the approach for solving the CEP development problem was introduced. Within this framework, a novel policy-based planning technique was coined. This technique extends existing planning techniques and provides means for different people to specify requirements to the planning processes independently. This is enabled by the utilisation of the XACML policy language, which was selected for the specification of these requirements within the policy-based planning technique. In contrast to the overall CEP generation framework, the policy-based planning technique is based on problem-independent mechanisms and

can be used for solving problems in different problem areas.

In addition to the flexible approach for specification of requirements for the planning process using different policies that can be devised independently and can be applicable only in specific situations, the developed policy-based planning technique preserves the required level of control over this process. For this purpose, standard XACML policy specification mechanisms are utilised and novel extensions for this policy language were introduced, facilitating the specification and enforcement of obligations.

For the problem-independent policy-based planning technique, a planning environment specification was designed within which CEP generation problems can be solved. The overall approach of this study assumes that specific requirements for the CEPs development should reflect local educational regulations, routines and criteria for decision making, which are specified by different educators in different ways. Therefore, the specification of the overall CEP planning environment reflects only essential student mobility and general educational processes and requirements. With the aid of the policy-based planning, regulations in force in different domains can be specified as policies and enforced during the planning. According to the Bologna Process (BP), for the construction of a CEP, Learning objects (LObjs) (modules and Educational Programmes (EPs)) of different universities should be compared based on their credits, educational levels and learning outcomes. In the developed technique, for the comparison of educational levels from different qualification frameworks and credit values given in credit units adopted in different domains, a transformation rules mechanism is utilised. For the learning outcomes comparison, a specialised similarity measure was adopted. Again, threshold values for this similarity measure and similarity measures derived from it are specified in policies by different policy authors at their discretion and are checked during the CEP development.

Possible planning performance improvements for the policy-based planning were also investigated in this thesis. Specifically, the planning-time policy enforcement peculiarities and specific characteristics of the CEP development problem were explored as sources of the performance improvements. Based on this, the advanced policy enforcement mechanism for the policy-based planning, viz., the postponed policy enforcement, was designed. For the CEP generation planning problem, a dedicated performance improvement technique, namely, the descending policy enforcement, was developed relying on the postponed policy enforcement.

10.2 Original contributions

CEP development framework

First of all, the novel CEP development framework was proposed for the generation of CEPs based on existing EPs and modules in an environment with heterogeneous regulations, governing different aspects of the educational process. Within the framework, the CEP development approach was

introduced and roles of different technologies, models and users in it were defined. The task of new CEPs generation has not been considered before within the e-Learning field, so it constitutes the novel approach for the student mobility processes support using computer technologies.

Policy-based planning technique

The policy-based management approach was applied to a new application area, namely, the problem-independent Hierarchical Task Network (HTN) planning, and the policy-based planning technique was designed. This provided the possibility to carry out planning in environments with heterogeneous regulations, supported by different people independently. These regulations are specified as policies using the attribute-based policy specification language. Different policies can be applicable in different situations or they can specify requirements to the same process, but from different perspectives. The independently specified policies are consistently enforced during the planning, so the resulting plan conforms with all policies applicable to it.

Obligations validation mechanism. In the policy-based planning technique, policies should be specified and enforced in a controlled manner. Procedures for the authorisation decisions processing and conflict resolution between them can be defined using the standard XACML policy language constructs. As analogous mechanisms for the XACML policy obligations are absent, the XACML obligation specification mechanism was extended and a novel obligations validation mechanism was proposed. Using obligations validation rules, it is possible to determine which obligations can be produced in specific situations and define a required order of their execution. In addition to the validation of the whole obligations set produced by the policy engine, it is possible to validate obligations generated by specific policies separately. That is, an author of a composite policy can specify validation rules that determine which obligations can be produced by a specific subordinate policy.

Adaptive policy requests construction. During the policy-based planning, for each planning action a policy evaluation request should be generated that contains the specification of action and information from the planner's world state about the objects that this action refers to. As in the XACML policy language there is no mechanism to determine which information in the policy request will be required during the policy evaluation at the moment of request creation, the adaptive policy requests construction procedure was designed in Chapter 5. Using it, policies are analysed at the time of their specification. Based on the information extracted at this stage, during the policy request generation special constructs called object contexts are created and placed into the policy request. These contexts contain all information about the object that can be required during the policy evaluation.

Formalisation of the CEP generation problem as a planning task

For solving the CEP generation problem using the policy-based planner, the corresponding planning environment was specified and the CEP generation problem was formalised as a planning task in this environment. For this purpose, CG techniques, which are used to solve a similar problem for non-mobile EPs, were adapted to the student mobility problem area. The CEP generation problem has a number of distinctive characteristics that were incorporated into the CEP generation planning environment specification. For example, there is support for different student mobility scenarios, utilisation of EPs and modules from different education providers, support of educational modules recognition as an integral part of the programme design, and others. For the representation of developed CEPs to the users, CEP process model was designed. It describes the educational process carried out when a student is studying according to a CEP in a detailed and comprehensible form.

Planning performance improvement techniques

Within this study, two planning performance improvement techniques were designed for policy-based planning by the means of enhancements introduced into the process of policy enforcement during the planning. The principle that the planning performance can be improved if future dead-ends can be detected earlier during the planning was applied to the policy-based planning and resulted in the development of the **postponed policy enforcement** mechanism. In this mechanism, dead-ends can be detected when policies are evaluated at earlier stages of the planning. For this purpose, policy evaluation requests corresponding to future actions are generated and evaluated during the planning. If during the evaluation of these requests a standard XACML policy decision can not be inferred due to an absence of required information, such requests should be postponed. During the planning, such policy requests are refined when new information arises for them and re-evaluated based on this information.
The postponed policy enforcement mechanism was applied to the CEP generation problem and the **descending policy evaluation technique** was designed. This performance improvement technique optimises the LObj selection process during the CEP construction. Known information about the LObj is used for the policy evaluation and is incrementally refined during the planning. Additionally, this technique relies on the hierarchical multidomain structure of the considered planning environment. Policies for different domains are evaluated independently in descending order, gradually limiting the scope for the LObj selection.

Partial policy evaluation for XACML policy language

The partial policy evaluation for the XACML policy language was introduced in Chapter 6 as an extension of the standard XACML policy request specification technique and its policy evaluation algorithm. It solves the problem of policy evaluation requests specification and their evaluation when some part of the information about the planning action is absent but it is known that it

will be provided in future. For this purpose, a new Indeterminate Temporal policy decision was introduced to designate a case when this missing information prevents a standard XACML policy decision inference. The partial policy evaluation was designed relying on the formal model of the XACML policy language, introduced in Chapter 4.

10.3 Revisiting success criteria

To answer the research questions and satisfy the success criteria, defined in Chapter 5.1, the CEP generation technique was developed. It was initially presented in Chapter 3, where the CEP generation framework and the overall CEP development process were introduced. In the subsequent chapters, it was refined and specified as a planning problem within the specially designed planning environment. Finally, its feasibility has been shown using the case studies in Chapter 9, which exploited educational problems characterised by different scales and involving different mobility scenarios.

As users of this technique can have different requirements to the CEPs being developed, a specialised mechanism for specification of diverse user requirements was envisaged in the CEP generation framework. This mechanism was implemented as a part of the planning environment formalisation in Chapter 7. According to it, requirements to the CEP can be specified from three different perspectives. Structure requirements are defined as a Basic track (BTr) of the future CEP, process requirements are specified using an initial task network and properties requirements are specified as constraints on the CEP properties. For provision of the additional flexibility during the CEP requirements specification, users can utilise compound tasks from different levels of the task network and entities situated at different levels of property hierarchies (e.g., the domains tree). Another distinctive feature of the CEP development is a requirement to deal with mobility scenarios and physical movements of the student. In the technique developed, based on the specified CEP requirements, that is, on the BTr and high-level specification of the educational process, possible student mobility scenarios are generated. For this purpose, HTN decomposition methods representing different basic student mobility schemas are utilised. These methods can be applied in a nested manner and produce arbitrary complex composite mobility scenarios that satisfy the student's requirements.

The heterogeneity of regulations governing the CEP generation has led to the development of the novel policy-based planning technique. These regulations are formalised using the XACML policy language providing the means to independently specify simple policies that are applicable only in certain situations and combine them into more complex policy sets. The policy-based planning technique, designed in Chapter 5, provides means to control the planning process based on the XACML policies specified. Authorisation policies are used to restrict the applicability of actions during the planning; obligations are used to extend the decomposition methods with additional

actions in order to conform to the routines established. In order to provide different authors with a tool for independent policy specification, the policy scoping and combining mechanisms of the XACML policy language are utilised. They are used to define when certain policies can be applied and establish the procedures for processing of decisions returned by different policies. Using the novel obligations validation mechanism designed for XACML in this study, it is possible to define a set of possible obligations that can be returned by different policies and even specify valid combinations of obligation actions (see Chapter 5).

Finally, for improvement of the planning performance, the postponed policy enforcement mechanism was designed as an extension of the policy-based planning in Chapter 6. The performance gains are caused by the improvements introduced into the planning-time policy enforcement. The descending policy evaluation technique, designed in Chapter 7, applies this mechanism to the CEP generation planning problem and relies on the specific characteristics of the CEP development planning environment. In Chapter 9, the performance gains produced by different versions of the descending policy evaluation were evaluated and the impacts of different policies and planning environment characteristics were analysed in four series of experiments. It was shown that this technique is advantageous and leads to a decrease in CPU time in environments with a high policy stringency[1] and environments where permanent policy decisions can be produced during policy evaluation for higher level domains. The performance of the descending policy evaluation also depends on the shape of the planning environment domain structure. The descending policy evaluation is based on the possibility of evaluating policies jointly for all EPs within some domain, so the descending policy evaluation shows better results in environments with a large number of EPs. On the other hand, this technique failed to bring performance gains in environments with high fragmentation, since high values of domain tree's branching factor lead to superfluous evaluations of policies in supplementary domains.

10.4 Future work

Obviously, there are many possibilities for continuation and extension of the presented research. Some of the possible future extensions are as follows:

Enhancement of the CEP generation planning environment specification

Now only the pure credits mobility CEP type and only one-to-one modules recognition is supported in the designed CEP generation planning environment. Hence, while keeping the same approach to for the educational and mobility processes modelling, the specification of this planning environment can be extended to include other CEP types, for example, double and joint diploma degrees, more possibilities for recognition or a virtual mobility support, when a student physically is studying at

[1] That is, with a high ratio of Deny and Permit decisions produced.

his (or her) home university but takes online or blended modules at other universities via the VLE technology.

Optimisation of modules-based CEP generation phase

Another interesting direction is performance improvement for lower level CEP generation processes, operating at the modules level. The descending policy evaluation technique, which was developed, covers only higher-level stages of the CEP construction, that is, the BTr construction. However, lower level processes of the detailed CEP development are also computationally intensive, as they involve solving several interrelated constraints satisfaction problems: recognition of modules, optional modules selection and modifications of the modules sets in semesters. In this study, the development of detailed CEP process models was implemented using the trial-and-error approach. In the case studies, these processes were tried for a small-scale problem, involving only two EPs, but required significant computational time. In future, these processes should be analysed in more detail and a more advanced approach should be designed, for example, based on existing techniques in the constraints satisfaction or truth-maintenance areas.

Enhancement of user interface and user-interaction processes

Now, the CEP generation system is implemented as a prototype without a GUI and with elementary user interactions processes. The specification of planning problems to be solved should be done externally in a file and pre-compiled so that they can be used by the tool. A GUI that can be developed for this system should provide convenient facilities for the CEP requirements specification and support visual representation of the developed CEPs. The CEP requirements specification phase can be implemented as a dialog with the user, supporting intermediate consistency checks and providing hints. Another issue arising when system-user interactions are considered is the requirement for an analytical functionality, so that a user can compare and analyse CEPs and BTrs developed by the system. This issue was touched on in Chapter 6, when the CEP construction procedure was described. As a solution, a user could be provided with a set of metrics that he (or she) can use for the analysis and comparison of different CEP and BTrs produced. Correspondingly, based on the comparison results, the user can select the required CEP or update input requirements.

Development of a combined descending policy evaluation technique

As was discussed during the performance analysis, different versions of the descending policy evaluation (i.e., ordered or unordered) give better results depending on specific characteristics of the policies and planning environment. Moreover, these characteristics can vary in different areas of the planning environment being used. Therefore, a combined technique is required that supports both versions of the descending policy evaluation. Using this technique, it could be possible to

analyse the planning environment and, based on its properties, apply more efficient version of the descending policy evaluation (moreover, machine learning methods can be used for this derivation).

Application of the policy-based planning in a new problem area

As was mentioned, the core policy-based planning technique is a problem-independent planning technique. Therefore, an interesting question that arises is how it behaves in other problem domains. The motivating factor for choosing the policy-based planning as a tool for solving a planning problem is the existence of heterogeneous requirements that should be taken into account during the planning and that should be authored by different people. The scope of applications for automated planning technologies is being constantly extended, and new application areas are appearing, so more challenging tasks for planning, where planning is applied in a global context, will appear. One such area, which was identified as promising for the application of planning along with the policy-based management techniques, is a scientific workflow generation. Planning technologies are successfully utilised for the construction of automatic workflow generation systems. However, these workflows are usually implemented for large scale systems, utilising resources of different scientific organisation, for example, using a grid infrastructure [72]. Requirements for usage of these resources are determined by the resource owner independently but they should be coherently taken into account when a specific workflow is planned. Therefore, policy-based planning can be used during the scientific workflow development to ease the problem of specification and enforcement of these requirements.

Bibliography

[1] S. Adam, “Improving the recognition system of degrees and study credit points in the european higher education area,” University of Latvia, Riga, Tech. Rep., 2004.

[2] ——, “Learning outcomes current developments in Europe: update on the issues and applications of learning outcomes associated with the Bologna Process,” in *Proceedings of Edinburgh Bologna Seminar: Learning Outcomes Based Higher Education - The Scottish Experience*, 2008.

[3] C. Aldrich, *Simulations and the future of learning: an innovative (and perhaps revolutionary) approach to e-learning.* Pfeiffer, 2005.

[4] V. Aleven and K. Koedinger, “Limitations of student control: Do students know when they need help?” in *Intelligent Tutoring Systems*, ser. Lecture Notes in Computer Science, G. Gauthier, C. Frasson, and K. VanLehn, Eds., vol. 1839. Springer Berlin / Heidelberg, 2000, pp. 292–303.

[5] P. Alves, L. Amaral, and J. Pires, “Case-based reasoning approach to Adaptive Web-Based Educational Systems,” in *Proceedigns of 8th IEEE International Conference on Advanced Learning Technologies (ICALT ’08)*, 2008, pp. 260–261.

[6] A. Anderson, “A comparison of two privacy policy languages: EPAL and XACML,” in *Proceedings of the 3rd ACM workshop on Secure web services (SWS ’06).* New York, NY, USA: ACM, 2006, pp. 53–60.

[7] N. Aziah, “Design of a motivational scaffold for the malaysian e-Learning environment.” *Educational Technology & Society*, vol. 15, no. 1, pp. 137–151, 2012.

[8] S. Babic, “E-learning environment compared to traditional classroom,” in *Proceedings of the 34th International Convention MIPRO*, 2011, pp. 1299–1304.

[9] F. Bacchus and F. Kabanza, “Using temporal logic to control search in a forward chaining planner,” in *New directions in AI planning*, M. Ghallab and A. Milani, Eds. Amsterdam, The Netherlands: IOS Press, 1996, pp. 141–153.

[10] ——, “Using temporal logics to express search control knowledge for planning,” *Artificial Intelligence*, vol. 116, pp. 123–191, 2000.

[11] J. Baier, C. Fritz, M. Bienvenu, and S. McIlraith, "Beyond Classical Planning: Procedural Control Knowledge and Preferences in State-of-the-art Planners," in *Proceedings of the 23rd National Conference on Artificial Intelligence (AAAI'08)*, vol. 3. AAAI Press, 2008, pp. 1509–1512.

[12] M. Baldoni, C. Baroglio, N. Henze, and V. Patti, "Setting up a framework for comparing Adaptive Educational Hypermedia: First steps and application on Curriculum Sequencing," in *Proceedings of ABIS-Workshop 2002: Personalization for the Mobile World, Workshop on Adaptivity and User Modeling in Interative Software Systems*, 2002, pp. 43–50.

[13] M. Baldoni, C. Baroglio, and V. Patti, "Applying logic inference techniques for gaining flexibility and adaptivity in tutoring systems," in *Proceedings of the 10th International Conference on Human-Computer Interaction (HCII'03)*, 2003, pp. 517–521.

[14] C. Barcelos, J. Gluz, and R. Vicari, "An agent-based federated learning objects search service," *Interdisciplinary Journal of E-Learning and Learning Objects*, vol. 7, 2011.

[15] S. Baskerville, F. MacLeod, and N. Saunders, "A Guide to UK Higher Education and Partnerships for Overseas Universities," UK Higher Education International and Europe Unit, Tech. Rep., 2011.

[16] M. Becker, C. Fournet, and A. Gordon, "SecPAL: design and semantics of a decentralized authorization language," Microsoft Research, Tech. Rep., 2006.

[17] *Framework for Qualifications of The European Higher Educational Area*, Bologna Working Group on Qualifications Frameworks Std., 2005.

[18] P. Bonatti and D. Olmedilla, "Rule-based policy representation and reasoning for the semantic web," in *Reasoning Web*, ser. Lecture Notes in Computer Science, G. Antoniou, U. Abmann, C. Baroglio, S. Decker, N. Henze, P.-L. Patranjan, and R. Tolksdorf, Eds. Springer Berlin / Heidelberg, 2007, vol. 4636, pp. 240–268.

[19] P. Bonatti, S. De Capitani di Vimercati, and P. Samarati, "An algebra for composing access control policies," *ACM Transactions on Information and System Security (TISSEC)*, vol. 5, no. 1, pp. 1–35, 2002.

[20] J. Bradshaw, P. Beautement, M. Breedy, L. Bunch, S. Drakunov, P. Feltovich, R. Hoffman, R. Jeffers, M. Johnson, S. Kulkarnt, J. Lott, A. Raj, N. Suri, and A. Uszok, "Making agents acceptable to people," in *Intelligent Technologies for Information Analysis: Advances in Agents, Data Mining, and Statistical Learning*, ser. Lecture Notes in Computer Science, N. Zhong and J. Liu, Eds. Springer Berlin / Heidelberg, 2004, pp. 361–400.

[21] S. Britain and O. Liber, "A framework for pedagogical evaluation of Virtual Learning Environments," Bangor University Centre for Learning Technology, Tech. Rep., 1999.

[22] A. Bronstein and C. Talcott, "String-functional semantics for formal verification of synchronous circuits," Department of Computer Science, Stanford University, CA, USA, Tech. Rep., 1988.

[23] P. Brusilovsky and C. Peylo, "Adaptive and Intelligent Web-based Educational Systems," *International Journal of Artificial Intelligence in Education*, vol. 13, pp. 156 – 169, 2003.

[24] P. Brusilovsky, "Adaptive and intelligent technologies for web-based education," in *Special Issue on Intelligent Systems and Teleteaching, Kunstliche Intelligenz*, 1999, pp. 19–25.

[25] P. Brusilovsky and J. Vassileva, "Course sequencing techniques for large-scale web-based education," *International Journal of Continuing Engineering Education and Lifelong Learning*, vol. 13, pp. 75–94, 2003.

[26] T. Bylander, "Complexity results for planning," in *Proceedings of the 12th International Joint Conference on Artificial intelligence (IJCAI'91).* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1991, pp. 274–279.

[27] S. Calo and J. Lobo, "A basis for comparing characteristics of policy systems," in *Proceedings of 7th IEEE International Workshop on Policies for Distributed Systems and Networks (Policy'06)*, Jun. 5–7, 2006, pp. 10–194.

[28] N. Capuano, M. Marsella, and S. Salerno, "ABITS: An Agent Based Intelligent Tutoring System for distance learning," in *Proceedings of the International Workshop in Adaptative and Intelligent Web-based Educational Systems*, 2000, pp. 17–28.

[29] J. Carbonell, O. Etzioni, Y. Gil, R. Joseph, C. Knoblock, S. Minton, and M. Veloso, "PRODIGY: An Integrated Architecture for Planning and Learning," *SIGART Bulletin*, vol. 2, no. 4, pp. 51–55, Jul. 1991.

[30] R. Chadha and L. Kant, *Policy-Driven Mobile Ad hoc Network Management.* Wiley-IEEE Press, 2008.

[31] C. Cheng and J. Yen, "Virtual Learning Environment (VLE): a Web-based collaborative learning system," in *Proceedings of the 31st Hawaii International Conference on System Sciences*, vol. 1, 1998, pp. 480–491.

[32] E. Chernikova, "A novel process model-driven approach to comparing educational courses using ontology alignment," Ph.D. dissertation, Faculty of Technology, De Montfort University, Leicester, UK, 2014.

[33] J. Clark and S. DeRose, *XML Path Language (XPath). Version 1.0*, World Wide Web Consortium (W3C) Std., 1999.

[34] B. Clement, E. Durfee, and A. Barrett, "Abstract reasoning for planning and coordination," *Journal of AI research*, vol. 28, pp. 453–515, 2007.

[35] R. Cosgrave, A. Risquez, T. Logan-Phelan, T. Farrelly, E. Costello, M. Palmer, C. McAvinia, N. Harding, and N. Vaughan, "Usage and uptake of Virtual Learning Environments in Ireland: Findings from a multi institutional study," *The All Ireland Journal of Teaching and Learning in Higher Education (AISHE-J)*, vol. 3, 2011.

[36] *The Common European Framework of Reference for Languages Learning, Teaching, Assessment*, Council of Europe Std., 1986.

[37] K. Currie and A. Tate, "O-plan: The open planning architecture," *Artificial Intelligence*, vol. 52, no. 1, pp. 49–86, 1991.

[38] N. Damianou, N. Dulay, E. Lupu, M. Sloman, and T. Tonouchi, "Tools for domain-based policy management of distributed systems," in *Proceeding of IEEE/IFIP Network Operations and Management Symposium (NOMS'02)*, 2002, pp. 203–217.

[39] N. Damianou, A. Bandara, M. Sloman, and E. Lupu, "A survey of policy specification approaches," Department of Computing, Imperial College of Science Technology and Medicine, Tech. Rep., 2002.

[40] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "Ponder: A language for specifying security and management policies for distributed systems," Imperial College of Science, Technology and Medicine, Tech. Rep., 2000.

[41] ——, "The Ponder policy specification language," in *Proceedings of International Workshop Policies for Distributed Systems and Networks (POLICY'01)*, ser. Lecture notes in Computer Science, M. Sloman, E. C. Lupu, and J. Lobo, Eds., vol. 1995. Springer Berlin / Heidelberg, 2001, pp. 18–38.

[42] J. De Coi and D. Olmedilla, "A review of trust management, security and privacy policy languages," in *Proceedings of International Conference on Security and Cryptography (SECRYPT'08)*, 2008.

[43] M. De Weerdt, A. Ter Mors, and C. Witteveen, "Multi-agent planning: An introduction to planning and coordination," Handouts of the European Agent Summer, Handouts, 2005.

[44] M. desJardins and M. Wolverton, "Coordinating a distributed planning system," *AI Magazine*, vol. 20, no. 4, pp. 45–53, 1999.

[45] G. Dodig-Crnkovic, "Scientific methods in computer science," in *Proceedings of Conference for Promotion of research in IT at new universities and university colleges in Sweden*, 2002.

[46] ——, "Constructive research and info-computational knowledge generation," in *Model-Based Reasoning in Science and Technology*, ser. Studies in Computational Intelligence, L. Magnani, W. Carnielli, and C. Pizzi, Eds. Springer Berlin Heidelberg, 2010, vol. 314, pp. 359–380.

[47] P. Doherty and J. Kvarnstrom, "TALplanner: A Temporal Logic-Based Planner," *AI Magazine*, vol. 22, no. 3, pp. 95–102, 2001.

[48] A. Dong and H. Li, "Ontology-based information integration in Virtual Learning Environment," in *Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence*, 2005, pp. 762–765.

[49] N. Dulay, E. Lupu, M. Sloman, and N. Damianou, "A policy deployment model for the ponder language," in *Proceedings of IEEE/IFIP International Symposium on Integrated Network Management (IM'01)*, 2001, pp. 14–18.

[50] E. Rissanen, Ed., *eXtensible Access Control Markup Language (XACML) Version 3.0*, OASIS (Organization for the Advancement of Structured Information Standards) Std., 22 January 2013.

[51] S. Edelkamp, S. Jabbar, and M. Nazih, "Large-Scale Optimal PDDL3 Planning with MIPS-XXL," in *Proceedings of the 5th International Planning Competition Booklet (IPC'06)*, 2006.

[52] S. Edelkamp and P. Kissmann, "GAMER: Bridging Planning and General Game Playing with Symbolic Search," in *Proceedings of the 5th International Planning Competition (IPC'08)*, 2008.

[53] R. Ellis, "Field guide to Learning Management Systems," Learning Circuits, American Society for Training & Development (ASTD), Guide, 2009.

[54] European Commission, "ECTS users' guide (European Credit Transfer and Accumulation System and the Diploma Supplement)," Directorate-general for Education and Culture, European Union, 2009.

[55] *The Bologna Declaration of 19 June 1999: Joint declaration of the European Ministers of Education*, European Higher Education Area Std., 1999.

[56] *Realising the European Higher Education Area*, European Higher Education Area Communique of the Conference of Ministers responsible for Higher Education, 2003.

[57] *Recommendation on mobility within the Community for students, persons undergoing training, young volunteers, teachers and trainers*, European Parliament, Council of the European Union Recommendation, 2001.

[58] *Recommendation (EC) No 2006/961 on transnational mobility within the Community for education and training purposes*, European Quality Charter for Mobility Std., 2006.

[59] European University Association, "Guidelines for quality enhancement in European joint master programmes," Erasmus Mundus Guidelines, 2006.

[60] *Mobility of students in Europe. Tertiary education. 2001 - 2012.*, Eurostat Std.

[61] R. Fikes and N. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," in *Proceedings of International Joint Conferences on Artificial Intelligence*, 1971, pp. 608–620.

[62] K. Fisler, S. Krishnamurthi, L. Meyerovich, and M. Tschantz, "Verification and change-impact analysis of access-control policies," in *Proceedings of the 27th Int. Conference on Software engineering (ICSE '05)*. New York, NY, USA: ACM, 2005, pp. 196–205.

[63] P. Forest, "Challenges and opportunities in developing joint study programmes and degrees. An introductory overview of key issues, solutions, and future developments," in *Proceedings of the Conference on Higher Education - A Key to Future Development in the High North*, 2009.

[64] H. Friedrich, "Joint degrees - a hallmark of the European Higher Education Area?" in *Proceeding of Official Bologna Seminar, Berlin.* Europe Unit Universities UK, 2006.

[65] D. Gabbay, *Elementary Logics: a procedural perspective.* Prentice Hall Europe, 1998.

[66] V. Gehmlich, "Recognition of Credits - Achievements and (Problems) Challenges - A Stocktaking Exercise," Fachhochschule Osnabruck, Report from Bologna Conference, Riga, 2004.

[67] V. Gehmlich, N. Gibbs, R. Markeviciene, T. Mitchell, G. Roberts, A. Siltala, and M. Steinmann, "A practical guide to designing degree programmes with integrated transnational mobility," German Academic Exchange Service (DAAD), Germany, Tech. Rep., 2008.

[68] K. Georgouli, "Virtual Learning Environments - an overview," in *Proceedings of the 15th Panhellenic Conference on Informatics (PCI'11)*, 2011, pp. 63–67.

[69] A. Gerevini, P. Haslum, D. Long, A. Saetti, and Y. Dimopoulos, "Deterministic planning in the Fifth International Planning Competition: PDDL3 and experimental evaluation of the planners," *Artificial Intelligence*, vol. 173, pp. 619 – 668, 2009.

[70] A. Gerevini and D. Long, "Plan constraints and preferences in PDDL3 - the language of the fifth international planning competition," Department of Electronics for Automation, University of Brescia, Tech. Rep., 2005.

[71] P. Gigiola, P. Giuseppe, and D. Rossi, "Intelligent tutoring system (an overview)," i-Tutor project, European Commission, Tech. Rep., 2012.

[72] Y. Gil, E. Deelman, J. Blythe, and C. Kesselman, "Artificial intelligence and grids: workflow planning and beyond," *IEEE Intelligent Systems, special issue on E-Science*, vol. 19, 2004.

[73] M. Goodrich and R. Tamassia, *Data structures and algorithms in Java*, ser. World wide series in computer science. Wiley, 1998.

[74] *Processes for engineering a system (EIA-632)*, Government Electronics and Information Technology Association Engineering Department Electronic Industries Allience Std., 1999.

[75] J. Hoffmann, "FF: the Fast-Forward Planning system," *AI Magazine*, vol. 22, no. 3, pp. 57–62, 2001.

[76] A. Hoshyar and R. Sulaiman, "Introduction to e-Learning infrastructure," in *Proceedings of the European Conference on e-Learning*, 2010.

[77] O. Ilghami, "Documentation for JSHOP2," Department of Computer Science, University of Maryland, Tech. Rep., 2006.

[78] *ISO/IEC 10181-3:1996 Information technology - Open Systems Interconnection - Security frameworks for open systems: Access control framework*, ISO/IEC Std., 1996.

[79] V. Jaligama and F. Liarokapis, "An Online Virtual Learning Environment for Higher Education," in *Proceeding of Third International Conference on Games and Virtual Worlds for Serious Applications (VS-GAMES'11)*, 2011, pp. 207–214.

[80] H. Janicke, A. Cau, F. Siewe, H. Zedan, and K. Jones, "A compositional event & time-based policy model," in *Proceedings of Policies for Distributed Systems and Networks (POLICY'06)*, 2006, pp. 173–182.

[81] M. Jenkins and J. Hanson, *E-learning Series: A guide for senior managers*, ser. LTSN Generic Centre e-Learning Series. York, England: Learning and Teaching Support Network Generic Centre, 2003, vol. 1.

[82] N. Jennings and M. Wooldridge, "Applications of intelligent agents," *Agent Technology: Foundations, Applications and Markets*, pp. 3 – 28, 1998, Springer-Verlag, New York.

[83] N. Jennings, K. Sycara, and M. Wooldridge, "A roadmap of agent research and development," *Autonomous Agents and Multi-Agent Systems*, vol. 1, no. 1, pp. 7–38, 1998.

[84] JISC infoNet, "Bologna process & strategic challenges," JISC infoNet InfoKit, 2009. [Online]. Available: http://tools.jiscinfonet.ac.uk/downloads/bologna/Bologna.pdf

[85] M. Johnson, P. Feltovich, J. Bradshaw, and L. Bunch, "Human-robot coordination through dynamic regulation," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA'08)*, 2008, pp. 2159–2164.

[86] S. Junor and A. Usher, "Student Mobility & Credit Transfer. A national and global survey," Educational Policy Institute, Tech. Rep., 2008.

[87] S. Kambhampati, "Design tradeoffs in partial order (plan space) planning," in *Proceedings of the 2nd International Conference on AI Planning Systems (AIPS'94)*, 1994, pp. 92–97.

[88] P. Karampiperis and D. Sampson, "Adaptive instructional planning using ontologies," in *Proceedings of IEEE International Conference on Advanced Learning Technologies*, 2004, pp. 126–130.

[89] ——, "Automatic learning object selection and sequencing in Web-Based Intelligent Learning Systems," in *Web-Based Intelligent e-Learning Systems: Technologies and Applications*, Z. Ma, Ed. Information Science Publishing, 2005, ch. 3, pp. 56–71.

[90] M. Kelo, U. Teichler, and B. Wachter, "EURODATA - Student Mobility in European Higher Education," Academic Cooperation Association (ACA), Tech. Rep., 2006.

[91] R. King, "International student mobility literature review," British Council, UK National Agency for Erasmus, Tech. Rep., 2007.

[92] J. Knight, "Doubts and dilemmas with double degree programs," Universitat Oberta de Catalunya, Barcelona, Tech. Rep., 2001.

[93] ——, "Joint and double degree programmes: Vexing questions and issues," The OBSERVATORY on borderless higher education, Tech. Rep., 2008.

[94] V. Kolovski, J. Hendler, and B. Parsia, "Analyzing web access control policies," in *Proceedings of the 16th International Conference on World Wide Web (WWW'07)*. New York, NY, USA: ACM, 2007, pp. 677–686.

[95] A. Kumar, R. Pakala, R. Ragade, and J. Wong, "The Virtual Learning Environment system," in *Proceedings of the 28th Annual Frontiers in Education Conference (FIE '98)*, vol. 2, 1998, pp. 711–716.

[96] J. Kvarnstrom and M. Magnusson, "TALplanner in the Third International Planning Competition: Extensions and control rules," *Journal of Artificial Intelligence Research*, vol. 20, pp. 343 – 377, 2003.

[97] *IEEE Standard for Learning Object Metadata 1484.12.1-2002*, Learning Technology Standards Committee Std., 2002.

[98] J. Levine, H. Westerberg, M. Galea, and D. Humphreys, "Evolutionary-based Learning of Generalised Policies for AI Planning Domains," in *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO '09)*. New York, NY, USA: ACM, 2009, pp. 1195–1202.

[99] N. Li, Q. Wang, W. Qardaji, E. Bertino, P. Rao, J. Lobo, and D. Lin, "Access control policy combining: theory meets practice," in *Proceedings of the 14th ACM symposium on Access control models and technologies (SACMAT'09)*. New York, NY, USA: ACM, 2009, pp. 135–144.

[100] K. Lukka, "The constructuve research approach," in *Case study research in logistics*. Turku School of Economics and Business Administration, 2003, vol. 1, pp. 83 – 101.

[101] E. Lupu and M. Sloman, "Conflicts in policy-based distributed systems management," *IEEE Transactions on Software Engineering*, vol. 25, no. 6, pp. 852–869, 1999.

[102] H. Ma, Z. Zheng, F. Ye, and S. Tong, "The applied research of cloud computing in the construction of collaborative learning platform under e-Learning environment," in *Proceedings of International Conference on System Science, Engineering Design and Manufacturing Informatization (ICSEM'10)*, vol. 1, 2010, pp. 190–192.

[103] H. Mahon, Y. Bernet, and S. Herzog, "Requirements for a Policy Management System," Internet Engineering Task Force (IETF), Tech. Rep., 1999.

[104] D. Mallon, J. Bersin, C. Howard, and K. O'Leonard, "Learning Management Systems (Executive Summary)," Bersin & Associates, Tech. Rep., 2009.

[105] B. Manville, "Organizing enterprise-wide e-learning and human capital management," *Chief Learning Officer Magazine*, 2003.

[106] M. Martín and H. Geffner, "Learning Generalized Policies from Planning Examples Using Concept Languages," *Applied Intelligence*, vol. 20, no. 1, pp. 9 – 19, Jan. 2004.

[107] R. Mazza and V. Dimitrova, "CourseVis: A graphical student monitoring tool for supporting instructors in web-based distance courses," *International Journal of Human-Computer Studies*, vol. 65, no. 2, pp. 125 – 139, 2007.

[108] A. Metcalfe, M. Snitzer, and J. Austin, "Virtual Adaptive Learning Architecture (VALA)," in *Proceedings of IEEE International Conference on Advanced Learning Technologies*, 2001, pp. 7–10.

[109] E. Meyen, R. Aust, J. Gauch, H. Hinton, R. Isaacson, S. Smith, and Meng Yew Tee, "e-Learning: A programmatic research construct for the future," *Journal of Special Education Technology*, vol. 17, no. 3, pp. 37 – 46, 2002.

[110] E. Milgrom, "MESSAGE: methodology for engineering systems of software agents. Final guidelines for the indentification of relevant problem areas where agent techology is appropriate," EURESCOM, Tech. Rep., 2001.

[111] C. Milligan, "Delivering staff and professional development using Virtual Learning Environments," JISC Technology Applications Programme, Tech. Rep., 1999.

[112] *Federal Law on Higher and After-university professional education from 22.08.1996 No. 125-FZ*, The Ministry of Education and Science Federal law, 1996.

[113] *Approval of recognition and equivalence establishment procedure in Russian Federation for foreign state educational documents*, Order No. 128 on 14.04.2009, Ministry of Education and Science Ministerial order, 2009.

[114] N. Minsky and V. Ungureanu, "Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems," *ACM Transactions on Software Engineering and Methodology (TOSEM'00)*, vol. 9, pp. 273–305, 2000.

[115] J. Muzio, T. Heins, and R. Mundell, "Experiences with reusable e-learning objects: from theory to practice," *The Internet and Higher Education*, vol. 5, no. 1, pp. 21 – 34, 2002.

[116] D. Nau, M. Ghallab, and P. Traverso, *Automated Planning: Theory & Practice.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004.

[117] D. Nau, O. Ilghami, U. Kuter, W. Murdock, D. Wu, and F. Yaman, "SHOP2: an HTN planning system," *Journal of Artificial Intelligence Research*, vol. 20, pp. 379 – 404, 2003.

[118] D. Nau, H. Munoz-avila, Y. Cao, A. Lotem, and S. Mitchell, "Total-order planning with partially ordered subtasks," in *Proceedings of the 17th Int. Joint Conference on Artificial Intelligence*, 2001, pp. 425–430.

[119] D. Nau, S. Smith, and K. Erol, "Control strategies in HTN planning: Theory versus practice," in *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI'98/IAAI'98).* AAAI Press, 1998, pp. 1127–1133.

[120] R. Neisse, P. Costa, M. Wegdam, and M. van Sinderen, "An information model and architecture for context-aware management domains," in *Proceedings of IEEE Workshop on Policies for Distributed Systems and Networks (POLICY'08)*, 2008, pp. 162–169.

[121] Q. Ni, E. Bertino, and J. Lobo, "D-algebra for composing access control policy decisions," in *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security (ASIACCS '09).* New York, NY, USA: ACM, 2009, pp. 298–309.

[122] S. Nickel, T. Zdebel, and D. Westerheijden, "Joint Degrees in European Higher Education. Obstacles and opportunities for transnational programme partnerships," EUREGIO, Centre for Higher Education development, Center for Higher Education Policy Studies, Tech. Rep., 2009.

[123] H. Nwana, "Intelligent tutoring systems: an overview," *Artificial Intelligence Review*, vol. 4, no. 4, pp. 251–277, 1990.

[124] T. Orzechowski, "The use of Multi-Agents' Systems in e-Learning platforms," in *Proceeding of Siberian Conference on Control and Communications (SIBCON'07)*, 2007, pp. 64–71.

[125] A. Patel, "A conceptual framework for Internet based Intelligent Tutoring Systems," *Knowledge Transfer*, vol. II, pp. 117–124, 1997.

[126] C.-I. Pena, J.-L. Marzo, and J.-L. de la Rosa, "Intelligent Agents in a Teaching and Learning Environment on the Web," in *Proceedings of IEEE International Conference on Advanced Learning Technologies (ICALT)*, 2002.

[127] J. Penberthy and D. Weld, "UCPOP: A sound, complete, partial-order planner for ADL," in *Proceedings of 3d International Conference on Knowledge Representation and Reasoning*, 1992.

[128] P. Polsani, "Use and abuse of reusable learning objects," *Journal of Digital Information*, vol. 3, no. 4, 2003.

[129] C. Powers and M. Schunter, Eds., *Enterprise Privacy Authorization Language (EPAL 1.2).* IBM Corporation, 2003.

[130] Quality Assurance Agency for Higher Education, "Guidelines on the accreditation of prior learning," UK QAA Guidance, Quality Assurance Agency for Higher Education, UK, 2004.

[131] ——, "Guidelines for preparing programme specifications," UK QAA Guidance, 2006.

[132] ——, "Higher education credit framework for England: Guidance on academic credit arrangements in Higher Education in England," UK QAA Guidance, 2008.

[133] E. Remolina, S. Ramachandran, D. Fu, R. Stottler, and W. Howse, “Intelligent simulation-based tutor for flight training,” in *Proceedings of the Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC)*, 2004.

[134] C. Ribeiro, A. Zuquete, P. Ferreira, and P. Guedes, “SPL: An access control language for security policies with complex constraints,” in *Proceedings of the Network and Distributed System Security Symposium*, 1999, pp. 89–107.

[135] *Federal State Educational Standard of Russian Federation for Higher Professional Education in direction 230100 ”Informatics and Computing” (master)*, Russian Federation Ministry of Education and Science Std., 2009.

[136] E. Sacerdoti, “The nonlinear nature of plans,” in *Proceedings of the 4th International Joint Conference on Artificial intelligence*, vol. 1. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1975, pp. 206–214.

[137] A. Sangra, D. Vlachopoulos, and N. Cabrera, “Building an inclusive definition of e-learning: an approach to the conceptual framework,” *The International Review of Research in Open and Distance Learning*, vol. 13, no. 2, 2012.

[138] SCQF Partnership, “Scottish Credit and Qualifications framework (SCQF) Handbook: User guide,” SCQF Guide.

[139] B. Semkin, “On the analysis of sets of different sizes in the comparative floristics,” in *Proceedings of Komarov’s Reedings Conference’09*, no. 56, 2009, pp. 170–185.

[140] Y. Shoham, “An overview of agent-oriented programming,” in *Software agents*, J. Bradshaw, Ed. Cambridge, MA, USA: MIT Press, 1997, ch. An overview of agent-oriented programming, pp. 271–290.

[141] M. Sloman, “Policy driven management for distributed systems,” *Journal of Network and Systems Management*, vol. 2, pp. 333–360, 1994.

[142] M. Smith, A. Schain, K. Clark, A. Griffey, and V. Kolovski, “Mother, may I? OWL-based policy management at NASA,” in *Proceedings of OWL: Experiences and Directions Workshop (OWLED’07)*, ser. CEUR Workshop Proceedings, C. Golbreich, A. Kalyanpur, and B. Parsia, Eds., vol. 258, 2007.

[143] M. Steinmann, “Mobility and the Bologna Process,” German Academic Exchange Service (DAAD), Germany, Tech. Rep., 2008.

[144] B. Stepien, A. Felty, and S. Matwin, "A non-technical user-oriented display notation for XACML conditions," in *E-Technologies: Innovation in an Open World*, ser. Lecture Notes in Business Information Processing, G. Babin, P. Kropf, and M. Weiss, Eds. Springer Berlin / Heidelberg, 2009, vol. 26, pp. 53–64.

[145] B. Stepien, S. Matwin, and A. Felty, "Advantages of a non-technical XACML notation in role-based models," in *Proceeding of the 9th Annual Conference on Privacy, Security and Trust (PST'09).* IEEE, 2011, pp. 193–200.

[146] G. Stone, B. Lundy, and G. Xie, "Network policy languages: A survey and a new approach," in *Network*, vol. 15. IEEE, 2001, pp. 10 – 21.

[147] S. Stoyanov, I. Ganchev, I. Popchev, M. O'Droma, and V. Valkanova, "Agent-oriented middleware for InfoStation-based mLearning intelligent systems," in *Proceeding of IEEE Conference of Intelligent Systems*, 2010, pp. 91–95.

[148] S. Stoyanov, V. Valkanova, I. Ganchev, and M. O'Droma, "An approach and architecture supporting context-aware provision of mLearning services," in *Proceeding of the 2nd International Conference on Mobile, Hybrid, and On-Line Learning (ELML'10)*, 2010, pp. 11–16.

[149] S. Suebnukarn and P. Haddawy, "COMET: A collaborative tutoring system for medical problem-based learning," *Intelligent Systems*, vol. 22, no. 4, pp. 70–77, IEEE, 2007.

[150] N. Suri, J. Bradshaw, A. Uszok, M. Breedy, M. Carvalho, P. Groth, R. Jeffers, M. Johnson, S. Kulkarni, J. Lott, M. Burstein, B. Benyo, and D. Diller, "Toward DAML-based policy enforcement for semantic data transformation and filtering in multi-agent systems," in *Proceedings of the 2nd international joint conference on Autonomous Agents and MultiAgent Systems (AAMAS'03).* New York, NY, USA: ACM, 2003, pp. 1132–1133.

[151] Sussex Centre for Migration Research, University of Sussex and the Centre for Applied Population Research, University of Dundee, "International student mobility," Issues paper, 2004.

[152] K. Sycara, "Multiagent systems," *AI Magazine*, vol. 19, pp. 79–92, 1998.

[153] T. Moses, Ed., *eXtensible Access Control Markup Language (XACML) Version 2.0*, OASIS (Organization for the Advancement of Structured Information Standards) Std., 1 February 2005.

[154] A. Tate and B. Drabble, "O-Plan2: Choice ordering mechanisms in an AI planning architecture," *Proceedings of the Workshop on Innovative Approaches to Planning*, pp. 192–197, 1990.

[155] A. Tate, "Generating project networks," in *Proceedings of the 5th international joint conference on Artificial intelligence*, vol. 2. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1977, pp. 888–893.

[156] R. Tsuneto, K. Erol, J. Hendler, and D. Nau, "Commitment strategies in Hierarchical Task Network planning," in *Proceedings of the Association for the Advancement of Artificial Intelligence Conference (AAAI/IAAI'96)*, vol. 1, 1996, pp. 536–542.

[157] R. Tsuneto, J. Hendler, and D. Nau, "Analyzing external conditions to improve the efficiency of HTN planning," in *Proceedings of the16th National Conference on Artificial Intelligence*, 1998, pp. 913–920.

[158] R. Tsuneto, D. Nau, and J. Hendler, "Plan-refinement strategies and search-space size," in *Proceedings of the 4th European Conference on Planning (ECP'97)*, ser. Lecture Notes in Artificial Intelligence, S. Steel and R. Alami, Eds., vol. 1348. Springer Berlin / Heidelberg, 1997, pp. 414–426.

[159] *Code of practice for the assurance of academic quality and standards in Higher Education. Collaborative provision and flexible and distributed learning (including e-learning)*, The UK Quality Assurance Agency for Higher Education Std., 2010.

[160] *UK Quality Code for Higher Education. Part A: Setting and maintaining threshold academic standards*, The UK Quality Assurance Agency for Higher Education Std., 2011.

[161] *UK Quality Code for Higher Education. Part B: Assuring and enhancing academic quality*, The UK Quality Assurance Agency for Higher Education Std., 2011.

[162] C. Ullrich, "Course Generation based on HTN planning," in *Proceedings of the 13th Annual Workshop of the SIG Adaptivity and User Modeling in Interactive Systems*, 2005, pp. 74–79.

[163] C. Ullrich and E. Melis, "Pedagogically founded courseware generation based on HTN-planning," *Expert Systems and Applications*, vol. 36, no. 5, 2009.

[164] J. Underwood and R. Luckin, "What is AIED and why does Education need it?" Artificial Intelligence in Education, Teaching and Learning Research Programme, UK, Tech. Rep., 2011.

[165] E. Unit, "The future of UK student mobility," Europe Unit, Universities UK, Tech. Rep., 2008.

[166] *International Standard Classification of Education (ISCED)*, United Nations Educational, Scientific and Cultural Organization (UNESCO) Institute for Statistics Std., 2011.

[167] A. Uszok, J. Bradshaw, P. Hayes, R. Jeffers, M. Johnson, S. Kulkarni, J. Lott, and L. Bunch, “DAML reality check: A case study of KAoS domain and policy services,” in *Proceedings of the International Semantic Web Conference (ISWC’03)*, 2003.

[168] A. Uszok, J. Bradshaw, R. Jeffers, N. Suri, P. Hayes, M. Breedy, L. Bunch, M. Johnson, S. Kulkarni, and J. Lott, “KAoS Policy and Domain Services: Toward a description-logic approach to policy representation, deconfliction, and enforcement,” in *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY’03)*. Washington, DC, USA: IEEE Computer Society, 2003, pp. 93–97.

[169] J. Vassileva, “Reactive instructional planning to support interacting teaching strategies,” in *Proceedings of the 7th World Conference on AI and Education (AI-ED’95)*, Washington, 1995.

[170] ——, “An architecture and methodology for creating a domain-independent plan-based intelligent tutoring system,” *Education and Training Technology International*, vol. 27, pp. 386–397, 1990.

[171] ——, “Dynamic courseware generation: at the cross point of CAL, ITS and Authoring,” in *Proceedings International Conference on Computers in Education*, 1995.

[172] ——, “DCG+GTE: Dynamic courseware generation with teaching expertise,” *Instructional Science*, vol. 26, pp. 317–332, 1998.

[173] ——, “TOBIE: An implementation of a domain- independent ITS-architecture in the domain of symbolic integration,” *New Media and Telematic Technologies for Education in Eastern European Countries*, pp. 241–258, Twente University Press, 1997.

[174] J. Vassileva and B. Wasson, “Instructional planning approaches: from tutoring towards free learning,” in *Proceedings of EuroAIED’96*, 1996, pp. 1–8.

[175] R. Viccari, D. Ovalle, B. Jovani, and A. Jimenez, “ALLEGRO: Teaching/Learning Multi-Agent Environment using Instructional Planning and Cases-Based Reasoning (CBR),” *CLEI Electronical Journal*, vol. 10, no. 1, 2007. [Online]. Available: http://dblp.uni-trier.de/db/journals/cleiej/cleiej10.html#ViccariOJ07

[176] A. Vinha, “Reusable learning objects: theory to practice,” *Proceedings of the 10th Annual Conference on Innovation and Technology in Computer Science Education (SIGCSE’2005)*, vol. 37, no. 3, pp. 413–413, 2005.

[177] R. Wagenaar, "Bologna and the European Credit Transfer and Accumulation System (ECTS): role of Learning Outcomes and Workload in European perspective," Tuning project, Education and Culture DG, European Union, Tech. Rep., 2010.

[178] R. Walker, J. Voce, and J. Ahmed, "Survey of technology enhanced learning for higher education in the uk," Universities and Colleges Information Systems Association (UCISA), Tech. Rep., 2012.

[179] G. Weber, "Adaptive learning systems in the World Wide Web," in *Proceedings of the 7th International Conference on User modeling (UM'99).* Springer-Verlag New York, 1999, pp. 371–377.

[180] R. Wegener and J. Leimeister, "Do student-instructor co-created eLearning materials lead to better learning outcomes? Empirical results from a German Large Scale Course Pilot Study," in *Proceedings of th 45th Hawaii International Conference on System Science (HICSS)*, 2012, pp. 31–40.

[181] G. Weiss, Ed., *Multiagent Systems: Modern Approach to Distributed Artificial Intelligence.* Cambridge, MA, USA: MIT Press, 1999.

[182] R. Wies, "Using a classification of management policies for policy specification and policy transformation," in *Proceedings of the IFIP/IEEE International Symposium on Integrated network Management.* Chapman & Hall, 1995, pp. 44–56.

[183] D. Wilkins, "Can AI planners solve practical problems?" *Computational Intelligence*, vol. 6, no. 4, pp. 232–246, 1990.

[184] M. Yague, "Survey on XML-based Policy Languages for open environments," *Journal of Information Assurance and Security*, vol. 1, pp. 11 – 20, 2006.

[185] Q. Yang and A. Chan, "Delaying variable binding commitments in planning," in *Proceedings of the 2nd International Conference on AI Planning Systems (AIPS'94)*, 1994, pp. 182–187.

Appendix A

Evaluation of monotonicity for Partial policy evaluation algorithm

Monotonicity of Rule evaluation function for Partial policy evaluation $R^{ep} : EffectSpec \times TRVal^p \times TRVal^p \to M_2^p$

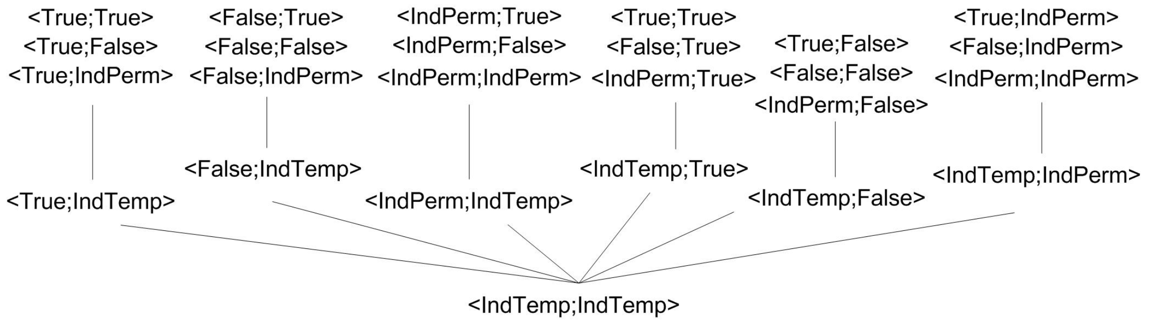

Figure A.1: Approximation order for $TRVal^p \times TRVal^p$ set

Each case above the line presents initial evaluation of R^{ep} function. Under the line, for this case each possible updates of R^{ep} arguments are considered, such that new values are more defined then initial argument values (according to the order in Figure A.1). At the bottom of the case, function results are compared according to the order defined on the M_2^p set. $\{P|D\}$ designates either Permit or Deny, depending on the value in the effect part of the rule considered.

Case 1:

$$\begin{array}{cccccccc}
R^{ep}: & IndTemp & \times & False & \times & \{P|D\} & \to & \{P|D\}(IndTemp) \\
\hline
R^{ep}: & True & \times & False & \times & \{P|D\} & \to & N/A \\
R^{ep}: & False & \times & False & \times & \{P|D\} & \to & N/A \\
R^{ep}: & IndPerm & \times & False & \times & \{P|D\} & \to & \{P|D\}(IndPerm) \\
\multicolumn{8}{c}{\{P|D\}(IndTerm) \sqsubseteq_{M_2^p} N/A, \{P|D\}(IndTerm) \sqsubseteq_{M_2^p} \{P|D\}(IndPerm)}
\end{array} \tag{A.1}$$

Case 2:

$$\begin{array}{lccccccccc} R^{ep}: & IndTemp & \times & IndPerm & \times & \{P|D\} & \rightarrow & \{P|D\}(IndTemp) \\ \hline R^{ep}: & True & \times & IndPerm & \times & \{P|D\} & \rightarrow & \{P|D\}(IndPerm) \\ R^{ep}: & False & \times & IndPerm & \times & \{P|D\} & \rightarrow & N/A \\ R^{ep}: & IndPerm & \times & IndPerm & \times & \{P|D\} & \rightarrow & \{P|D\}(IndPerm) \\ \multicolumn{8}{c}{\{P|D\}(IndTerm) \sqsubseteq_{M_2^p} N/A, \{P|D\}(IndTerm) \sqsubseteq_{M_2^p} \{P|D\}(IndPerm)} \end{array} \tag{A.2}$$

Case 3:

$$\begin{array}{lccccccc} R^{ep}: & IndTemp & \times & IndTemp & \times & \{P|D\} & \rightarrow & \{P|D\}(IndTemp) \\ \hline R^{ep}: & True & \times & True & \times & \{P|D\} & \rightarrow & \{P|D\} \\ R^{ep}: & True & \times & False & \times & \{P|D\} & \rightarrow & N/A \\ R^{ep}: & True & \times & IndPerm & \times & \{P|D\} & \rightarrow & \{P|D\}(IndPerm) \\ R^{ep}: & True & \times & IndTemp & \times & \{P|D\} & \rightarrow & \{P|D\}(IndTemp) \\ R^{ep}: & False & \times & * & \times & \{P|D\} & \rightarrow & N/A \\ R^{ep}: & IndPerm & \times & * & \times & \{P|D\} & \rightarrow & \{P|D\}(IndPerm) \\ R^{ep}: & IndTemp & \times & * & \times & \{P|D\} & \rightarrow & \{P|D\}(IndTerm) \\ \multicolumn{8}{c}{\{P|D\}(IndTerm) \sqsubseteq_{M_2^p} N/A, \{P|D\}(IndPerm), \{P|D\}(IndTerm), \{P|D\}} \end{array} \tag{A.3}$$

Case 4:

$$\begin{array}{lccccccc} R^{ep}: & IndTemp & \times & True & \times & \{P|D\} & \rightarrow & \{P|D\}(IndTemp) \\ \hline R^{ep}: & True & \times & True & \times & \{P|D\} & \rightarrow & \{P|D\} \\ R^{ep}: & False & \times & True & \times & \{P|D\} & \rightarrow & N/A \\ R^{ep}: & IndPerm & \times & True & \times & \{P|D\} & \rightarrow & \{P|D\}(IndPerm) \\ \multicolumn{8}{c}{\{P|D\}(IndTerm) \sqsubseteq_{M_2^p} N/A, \{P|D\}(IndPerm), \{P|D\}} \end{array} \tag{A.4}$$

Case 5:

$$\begin{array}{lccccccc} R^{ep}: & False & \times & IndTemp & \times & \{P|D\} & \rightarrow & N/A \\ \hline R^{ep}: & False & \times & False & \times & \{P|D\} & \rightarrow & N/A \\ R^{ep}: & False & \times & True & \times & \{P|D\} & \rightarrow & N/A \\ R^{ep}: & False & \times & IndPerm & \times & \{P|D\} & \rightarrow & N/A \\ \multicolumn{8}{c}{N/A \sqsubseteq_{M_2^p} N/A} \end{array} \tag{A.5}$$

Case 6:

$$\begin{array}{lcccccccc} R^{ep}: & IndPerm & \times & IndTemp & \times & \{P|D\} & \rightarrow & \{P|D\}(IndPerm) \\ \hline R^{ep}: & IndPerm & \times & False & \times & \{P|D\} & \rightarrow & \{P|D\}(IndPerm) \\ R^{ep}: & IndPerm & \times & True & \times & \{P|D\} & \rightarrow & \{P|D\}(IndPerm) \\ R^{ep}: & IndPerm & \times & IndPerm & \times & \{P|D\} & \rightarrow & \{P|D\}(IndPerm) \\ \multicolumn{8}{c}{\{P|D\}(IndPerm) \sqsubseteq_{M_2^p} \{P|D\}(IndPerm)} \end{array} \tag{A.6}$$

Case 7:

$$\begin{array}{lcccccccc} R^{ep}: & True & \times & IndTemp & \times & \{P|D\} & \rightarrow & \{P|D\}(IndTemp) \\ \hline R^{ep}: & True & \times & False & \times & \{P|D\} & \rightarrow & N/A \\ R^{ep}: & True & \times & True & \times & \{P|D\} & \rightarrow & \{P|D\} \\ R^{ep}: & True & \times & IndPerm & \times & \{P|D\} & \rightarrow & \{P|D\}(IndPerm) \\ \multicolumn{8}{c}{\{P|D\}(IndTerm) \sqsubseteq_{M_2^p} N/A, \{P|D\}(IndPerm), \{P|D\}} \end{array} \tag{A.7}$$

Monotonicity for Permit-overrides Rule combining operation for Partial policy evaluation $\bullet_{rp}^{PO} : M_2^p \times M_2^p \rightarrow M_2^p$

Case 1:

$$\begin{array}{lccccc} \bullet_{rp}^{PO}: & P(IndTemp) & \times & Permit & \rightarrow & Permit \\ \hline \bullet_{rp}^{PO}: & * & \times & Permit & \rightarrow & Permit \\ \multicolumn{6}{c}{Permit \sqsubseteq_{M_2^p} Permit} \end{array} \tag{A.8}$$

Case 2[1]:

$$\begin{array}{lccccc} \bullet_{rp}^{PO}: & P(IndTemp) & \times & P(IndTemp) & \rightarrow & P(IndTemp) \\ \hline \bullet_{rp}^{PO}: & * & \times & Permit & \rightarrow & Permit \\ \bullet_{rp}^{PO}: & P(IndPerm) & \times & P(IndTemp) & \rightarrow & P(IndTemp) \\ \bullet_{rp}^{PO}: & N/A & \times & P(IndTemp) & \rightarrow & P(IndTemp) \\ \bullet_{rp}^{PO}: & P(IndPerm) & \times & P(IndPerm) & \rightarrow & P(IndPerm) \\ \bullet_{rp}^{PO}: & N/A & \times & P(IndPerm) & \rightarrow & P(IndPerm) \\ \bullet_{rp}^{PO}: & N/A & \times & N/A & \rightarrow & N/A \\ \multicolumn{6}{c}{P(IndTemp) \sqsubseteq_{M_2^p} N/A, P(IndPerm), Permit, P(IndTemp)} \end{array} \tag{A.9}$$

[1] This case was evaluated considering $\bullet_{rp}^{PO}$ commutativity.

Case 3:

$$\begin{array}{lcccc}
\hline
\bullet^{PO}_{rp}: & P(IndPerm) & \times & P(IndTemp) & \rightarrow & P(IndTemp) \\
\hline
\bullet^{PO}_{rp}: & P(IndPerm) & \times & Permit & \rightarrow & Permit \\
\bullet^{PO}_{rp}: & P(IndPerm) & \times & P(IndPerm) & \rightarrow & P(IndPerm) \\
\bullet^{PO}_{rp}: & P(IndPerm) & \times & N/A & \rightarrow & P(IndPerm)
\end{array}$$
$$P(IndTemp) \sqsubseteq_{M_2^p} P(IndPerm), Permit \tag{A.10}$$

Case 4:

$$\begin{array}{lcccc}
\hline
\bullet^{PO}_{rp}: & Deny & \times & P(IndTemp) & \rightarrow & P(IndTemp) \\
\hline
\bullet^{PO}_{rp}: & Deny & \times & Permit & \rightarrow & Permit \\
\bullet^{PO}_{rp}: & Deny & \times & P(IndPerm) & \rightarrow & P(IndPerm) \\
\bullet^{PO}_{rp}: & Deny & \times & N/A & \rightarrow & Deny
\end{array}$$
$$P(IndTemp) \not\sqsubseteq_{M_2^p} N/A, , P(IndTemp) \sqsubseteq_{M_2^p} P(IndPerm), Permit \tag{A.11}$$

Case 5:

$$\begin{array}{lcccc}
\hline
\bullet^{PO}_{rp}: & D(IndTemp) & \times & P(IndTemp) & \rightarrow & P(IndTemp) \\
\hline
\bullet^{PO}_{rp}: & * & \times & Permit & \rightarrow & Permit \\
\bullet^{PO}_{rp}: & N/A & \times & P(IndTemp) & \rightarrow & P(IndTemp) \\
\bullet^{PO}_{rp}: & Deny & \times & P(IndTemp) & \rightarrow & P(IndTemp) \\
\bullet^{PO}_{rp}: & D(IndPerm) & \times & P(IndTemp) & \rightarrow & P(IndTemp) \\
\bullet^{PO}_{rp}: & D(IndTemp) & \times & P(IndPerm) & \rightarrow & P(IndPerm) \\
\bullet^{PO}_{rp}: & Deny & \times & P(IndPerm) & \rightarrow & P(IndPerm) \\
\bullet^{PO}_{rp}: & D(IndPerm) & \times & P(IndPerm) & \rightarrow & P(IndPerm) \\
\bullet^{PO}_{rp}: & N/A & \times & P(IndPerm) & \rightarrow & P(IndPerm) \\
\bullet^{PO}_{rp}: & D(IndTemp) & \times & N/A & \rightarrow & D(IndTemp) \\
\bullet^{PO}_{rp}: & Deny & \times & N/A & \rightarrow & Deny \\
\bullet^{PO}_{rp}: & D(IndPerm) & \times & N/A & \rightarrow & D(IndPerm) \\
\bullet^{PO}_{rp}: & N/A & \times & N/A & \rightarrow & N/A
\end{array}$$
$$P(IndTemp) \not\sqsubseteq_{M_2^p} Deny, D(IndPerm), D(IndTemp)$$
$$P(IndTemp) \sqsubseteq_{M_2^p} P(IndPerm), Permit, P(IndTemp) \tag{A.12}$$

Case 6:

$$\begin{array}{cccccc} \bullet_{rp}^{PO}: & D(IndPerm) & \times & P(IndTemp) & \rightarrow & P(IndTemp) \\ \hline \bullet_{rp}^{PO}: & D(IndPerm) & \times & Permit & \rightarrow & Permit \\ \bullet_{rp}^{PO}: & D(IndPerm) & \times & P(IndPerm) & \rightarrow & P(IndPerm) \\ \bullet_{rp}^{PO}: & D(IndPerm) & \times & N/A & \rightarrow & D(IndPerm) \end{array} \tag{A.13}$$
$$P(IndTemp) \not\sqsubseteq_{M_2^p} D(IndPerm), P(IndTemp) \sqsubseteq_{M_2^p} P(IndPerm), Permit$$

Case 7:

$$\begin{array}{cccccc} \bullet_{rp}^{PO}: & N/A & \times & P(IndTemp) & \rightarrow & P(IndTemp) \\ \hline \bullet_{rp}^{PO}: & N/A & \times & Permit & \rightarrow & Permit \\ \bullet_{rp}^{PO}: & N/A & \times & P(IndPerm) & \rightarrow & P(IndPerm) \\ \bullet_{rp}^{PO}: & N/A & \times & N/A & \rightarrow & N/A \end{array} \tag{A.14}$$
$$P(IndTemp) \sqsubseteq_{M_2^p} P(IndPerm), Permit, N/A$$

Case 8:

$$\begin{array}{cccccc} \bullet_{rp}^{PO}: & D(IndTemp) & \times & Permit & \rightarrow & Permit \\ \hline \bullet_{rp}^{PO}: & * & \times & Permit & \rightarrow & Permit \end{array} \tag{A.15}$$
$$Permit \sqsubseteq_{M_2^p} Permit$$

Case 9:

$$\begin{array}{cccccc} \bullet_{rp}^{PO}: & D(IndTemp) & \times & P(IndPerm) & \rightarrow & P(IndPerm) \\ \hline \bullet_{rp}^{PO}: & Deny & \times & P(IndPerm) & \rightarrow & P(IndPerm) \\ \bullet_{rp}^{PO}: & D(IndPerm) & \times & P(IndPerm) & \rightarrow & P(IndPerm) \\ \bullet_{rp}^{PO}: & N/A & \times & P(IndPerm) & \rightarrow & P(IndPerm) \end{array} \tag{A.16}$$
$$P(IndPerm) \sqsubseteq_{M_2^p} P(IndPerm)$$

Case 10:

$$\begin{array}{cccccc} \bullet_{rp}^{PO}: & D(IndTemp) & \times & Deny & \rightarrow & Deny \\ \hline \bullet_{rp}^{PO}: & Deny & \times & Deny & \rightarrow & Deny \\ \bullet_{rp}^{PO}: & D(IndPerm) & \times & Deny & \rightarrow & Deny \\ \bullet_{rp}^{PO}: & N/A & \times & Deny & \rightarrow & Deny \end{array} \tag{A.17}$$
$$Deny \sqsubseteq_{M_2^p} Deny$$

Case 11[2]:

$$\begin{array}{lcccc} \bullet_{rp}^{PO}: & D(IndTemp) & \times & D(IndTemp) & \rightarrow & D(IndTemp) \\ \hline \bullet_{rp}^{PO}: & Deny & \times & Deny & \rightarrow & Deny \\ \bullet_{rp}^{PO}: & D(IndPerm) & \times & Deny & \rightarrow & Deny \\ \bullet_{rp}^{PO}: & N/A & \times & Deny & \rightarrow & Deny \\ \bullet_{rp}^{PO}: & D(IndTemp) & \times & Deny & \rightarrow & Deny \\ \bullet_{rp}^{PO}: & D(IndPerm) & \times & D(IndPerm) & \rightarrow & D(IndPerm) \\ \bullet_{rp}^{PO}: & N/A & \times & D(IndPerm) & \rightarrow & D(IndPerm) \\ \bullet_{rp}^{PO}: & D(IndTemp) & \times & D(IndPerm) & \rightarrow & D(IndTemp) \\ \bullet_{rp}^{PO}: & D(IndTemp) & \times & N/A & \rightarrow & D(IndTemp) \\ \bullet_{rp}^{PO}: & N/A & \times & N/A & \rightarrow & N/A \\ \multicolumn{6}{c}{D(IndTemp) \sqsubseteq_{M_2^p} D(IndPerm), Deny, N/A, D(IndTemp)} \end{array} \tag{A.18}$$

Case 12:

$$\begin{array}{lcccc} \bullet_{rp}^{PO}: & D(IndPerm) & \times & D(IndTemp) & \rightarrow & D(IndTemp) \\ \hline \bullet_{rp}^{PO}: & D(IndPerm) & \times & Deny & \rightarrow & Deny \\ \bullet_{rp}^{PO}: & D(IndPerm) & \times & D(IndPerm) & \rightarrow & D(IndPerm) \\ \bullet_{rp}^{PO}: & D(IndPerm) & \times & N/A & \rightarrow & D(IndPerm) \\ \multicolumn{6}{c}{D(IndTemp) \sqsubseteq_{M_2^p} D(IndPerm), Deny} \end{array} \tag{A.19}$$

Case 13:

$$\begin{array}{lcccc} \bullet_{rp}^{PO}: & N/A & \times & D(IndTemp) & \rightarrow & D(IndTemp) \\ \hline \bullet_{rp}^{PO}: & N/A & \times & Deny & \rightarrow & Deny \\ \bullet_{rp}^{PO}: & N/A & \times & D(IndPerm) & \rightarrow & D(IndPerm) \\ \bullet_{rp}^{PO}: & N/A & \times & N/A & \rightarrow & N/A \\ \multicolumn{6}{c}{D(IndTemp) \sqsubseteq_{M_2^p} D(IndPerm), Deny, N/A} \end{array} \tag{A.20}$$

[2] This case was evaluated considering $\bullet_{rp}^{PO}$ commutativity.

Monotonicity of composition Permit-overrides Rule combining operation for Partial policy evaluation (revised for composition $f_p \circ \bullet_{rp}^{PO} : M_2^p \times M_2^p \rightarrow M_1^p$)

Case 4:

$$\begin{array}{ccccccccc} Deny & \times & P(IndTemp) & \xrightarrow{\bullet_{rp}^{PO}} & P(IndTemp) & \xrightarrow{f_p} & IndTemp \\ \hline Deny & \times & Permit & \xrightarrow{\bullet_{rp}^{PO}} & Permit & \xrightarrow{f_p} & Permit \\ Deny & \times & P(IndPerm) & \xrightarrow{\bullet_{rp}^{PO}} & P(IndPerm) & \xrightarrow{f_p} & IndPerm \\ Deny & \times & N/A & \xrightarrow{\bullet_{rp}^{PO}} & Deny & \xrightarrow{f_p} & Deny \\ \multicolumn{7}{c}{IndTemp \sqsubseteq_{M_1^p} Permit, IndPerm, Deny} \end{array} \tag{A.21}$$

Case 5:

$$\begin{array}{ccccccccc} D(IndTemp) & \times & P(IndTemp) & \xrightarrow{\bullet_{rp}^{PO}} & P(IndTemp) & \xrightarrow{f_p} & IndTemp \\ \hline * & \times & Permit & \xrightarrow{\bullet_{rp}^{PO}} & Permit & \xrightarrow{f_p} & Permit \\ N/A & \times & P(IndTemp) & \xrightarrow{\bullet_{rp}^{PO}} & P(IndTemp) & \xrightarrow{f_p} & IndTemp \\ Deny & \times & P(IndTemp) & \xrightarrow{\bullet_{rp}^{PO}} & P(IndTemp) & \xrightarrow{f_p} & IndTemp \\ D(IndPerm) & \times & P(IndTemp) & \xrightarrow{\bullet_{rp}^{PO}} & P(IndTemp) & \xrightarrow{f_p} & IndTemp \\ D(IndTemp) & \times & P(IndPerm) & \xrightarrow{\bullet_{rp}^{PO}} & P(IndPerm) & \xrightarrow{f_p} & IndPerm \\ Deny & \times & P(IndPerm) & \xrightarrow{\bullet_{rp}^{PO}} & P(IndPerm) & \xrightarrow{f_p} & IndPerm \\ D(IndPerm) & \times & P(IndPerm) & \xrightarrow{\bullet_{rp}^{PO}} & P(IndPerm) & \xrightarrow{f_p} & IndPerm \\ N/A & \times & P(IndPerm) & \xrightarrow{\bullet_{rp}^{PO}} & P(IndPerm) & \xrightarrow{f_p} & IndPerm \\ D(IndTemp) & \times & N/A & \xrightarrow{\bullet_{rp}^{PO}} & D(IndTemp) & \xrightarrow{f_p} & IndTemp \\ Deny & \times & N/A & \xrightarrow{\bullet_{rp}^{PO}} & Deny & \xrightarrow{f_p} & Deny \\ D(IndPerm) & \times & N/A & \xrightarrow{\bullet_{rp}^{PO}} & D(IndPerm) & \xrightarrow{f_p} & IndPerm \\ N/A & \times & N/A & \xrightarrow{\bullet_{rp}^{PO}} & N/A & \xrightarrow{f_p} & N/A \\ \multicolumn{7}{c}{IndTemp \sqsubseteq_{M_1^p} Permit, IndPerm, Deny, N/A, IndTemp} \end{array} \tag{A.22}$$

Case 6:

$$\begin{array}{ccccccccc} D(IndPerm) & \times & P(IndTemp) & \xrightarrow{\bullet_{rp}^{PO}} & P(IndTemp) & \xrightarrow{f_p} & IndTemp \\ \hline D(IndPerm) & \times & Permit & \xrightarrow{\bullet_{rp}^{PO}} & Permit & \xrightarrow{f_p} & Permit \\ D(IndPerm) & \times & P(IndPerm) & \xrightarrow{\bullet_{rp}^{PO}} & P(IndPerm) & \xrightarrow{f_p} & IndPerm \\ D(IndPerm) & \times & N/A & \xrightarrow{\bullet_{rp}^{PO}} & D(IndPerm) & \xrightarrow{f_p} & IndPerm \\ \multicolumn{7}{c}{IndTemp \sqsubseteq_{M_1^p} Permit, IndPerm} \end{array} \tag{A.23}$$

Appendix B

CEP generation planning domain specification

Planning tasks definitions for Basic track development and validation

$Degree(Student, Track, Award^{Req})$ task designates complete education of the student *Student* at an EP or CEP whose structure satisfies the ITr requirements (specified as $Track$) and which results in the award satisfying the requirements $Award^{Req}$.

$Start_Degree(Student, Track(1, N_e^{Sl})[, |EP|_{[1,m]}])$ task designates initial stage of the education according to a CEP designed for the student *Student*. It corresponds to the initial part of the track $Track(1, N_e^{Sl})$, where N_e^{Sl} is the number of the last slot covered by this task. At the beginning of this task, the student is admitted to an EP. At the end of this task, the student transfers permanently from it[1]. During the execution of this task, the student can change EPs several times. Optionally, this task can contain EP interval $|EP|_{[n,m]}$, which will be utilised as a basis to construct the CEP structure in this part of the track[2].

$Finish_Degree(Student, Track(N_s^{Sl}, N_e^{Sl}), Award^{Req})$ task designates the final stage of the education according to a CEP within the interval of the track from N_s^{Sl} to N_e^{Sl}, designed for the student *Student*. At the beginning of this task, the student transfers to an EP that he (or she) wants to graduate from (i.e., the permanent incoming transfer is carried out). At the end, the student graduates from this EP with a degree that satisfies the requirements $Award^{Req}$. During the execution of this task, the student can change EPs several times.

$Proceed_Degree(Student, Track(N_s^{Sl}, N_e^{Sl})[, |EP|_{[n,m]} or Award^{Req}])$ task designates the intermediate stage of the education according to a CEP within the interval of the track from N_s^{Sl} to N_e^{Sl}, designed for the student *Student*. At the beginning of this task, the student moves or returns to an EP that he (or she) is not intended to graduate from (i.e., the temporal incoming transfer is carried out). At the end, the student transfers from this EP (i.e., the permanent outgoing transfer is carried out). During the execution of this task, the student

[1] That is, the student will not return to this EP in the future.

[2] This parameter is specified for this task when it was generated during the planning, as a result of higher-level compound task decomposition, rather than it has been specified in the CEP process requirements Req^{Prop}. Then, some initial part of this EP interval (or the whole EP interval) should be studied at the first slot of this track. These rules are applicable for other compound tasks that have EP intervals in their term lists.

can change EPs several times.

$Start_Degree_Probation(Student, Track(1, N_e^{Sl})[, |EP|_{[1,m]}])$ task designates the initial stage of the education according to a CEP within the interval of the track from slot 1 to slot N_e^{Sl}, designed for the student *Student*. At the beginning of this task, the student is admitted to an EP that he (or she) wants to graduate from. At the end, the student temporarily transfers from it[3]. During the execution of this task, the student can change EPs several times.

$Start_Degree_str(Student, Track(1, N_e^{Sl})[, |EP|_{[1,m]}])$ is derived from *Start_Degree*. At the end of this task, the student transfers from the current EP, which he (or she) is not intended to graduate from, but he (or she) will return to it in the future.

$Start_Degree_ret(Student, Track(N_s^{Sl}, N_e^{Sl})[, |EP|_{[n,m]}])$ task designates the intermediate stage of the education according to a CEP within the interval of the track from N_s^{Sl} to N_e^{Sl}, designed for the student *Student*. At the beginning of this task, the student returns to an EP that he (or she) has studies before and that he (or she) is not intended to graduate from. At the end, the student transfers from it, but is going to return to it in the future.

$Start_Degree_fin(Student, Track(N_s^{Sl}, N_e^{Sl})[, |EP|_{[n,m]}])$ task designates the intermediate stage of the education according to a CEP within the interval of the track from N_s^{Sl} to N_e^{Sl}, designed for the student *Student*. At the beginning of this task, the student moves to an EP that he (or she) has studies before, but which he (or she) is not intended to graduate from. At the end, the student transfers from this EP (the student will not return to this EP in the future).

$Finish_Degree_fin(Student, Track(N_s^{Sl}, N_e^{Sl}), [, |EP|_{[n,m]} or Award^{Req}])$ is derived from *Finish_Degree*. At the beginning of this task, the student transfers to an EP that he (or she) has already studied before and that he (or she) wants to graduate from (without further transfers).

$Finish_Degree_str(Student, Track(N_s^{Sl}, N_e^{Sl})[, |EP|_{[n,m]}])$ is derived from *Finish_Degree*. At the end of this task, the student transfers from the EP temporarily (he (or she) will return to this EP in the future and will graduate from it).

$Finish_Degree_ret(Student, Track(N_s^{Sl}, N_e^{Sl})[, |EP|_{[n,m]}])$ task designates the intermediate stage of the education according to a CEP within the interval of the track from N_s^{Sl} to N_e^{Sl}, designed for the student *Student*. At the beginning of this task, the student moves to an EP that he (or she) has studied before and that he (or she) wants to graduate from. At the end, the student temporarily transfers from this EP.

$!admitP(Student, Track, 1, |EP|_{[n,m]})$ action designates admission of the student *Student* to the university at the EP interval $|EP|_{[n,m]}$. 'P' indicates that this student will graduate from this EP. The task is always executed in first slot.

[3] That is, he (or she) will return to this EP in the future.

$Effects^+$: $admitted(Student, Track, 1, EP, n, m)$, $assessed(Student,\ Lev_{Name},\ Lev_{Scale},\ Prov_{EP})$, $graduate(Student, Track, EP)$

$!admitT(Student, Track, 1, |EP|_{[n,m]})$ action designates admission of the student *Student* to the university at the EP interval $|EP|_{[n,m]}$. 'T' indicates that the student will not graduate from this EP. The task is always executed in first slot.

$Effects^+$: $admitted(Student, Track, 1, EP, n, m)$, $assessed(Student,\ Lev_{Name},\ Lev_{Scale},\ Prov_{EP})$, $not_graduate(Student, Track, EP)$.

$\&choose_modules(Student, Track, k, |EP|_{[n,m]})$ task is a compound action designating the procedure for the optional modules selection within the EP interval $|EP|_{[n,m]}$ for the student *Student* in slot n. This procedure is carried out during the decomposition of this compound action.

$\&study_interval(Student, Track, k, |EP|_{[n,m]})$ task is a compound action designating the procedure of studying according to $|EP|_{[n,m]}$ in slot k (by the student *Student*). This procedure is carried out during the decomposition of this compound action.

$Effects^+$: $history(Student, Track, k, EP, n, m)$.

$!graduate(Student, Track, k, |EP|_{[n,m]})$ action designates graduation of the student *Student* from the EP interval $|EP|_{[n,m]}$ in slot k.

$Effects^-$: $assessed(Student, Lev_{Name}, Lev_{Scale}, Prov_{EP})$.

$Effects^+$: $education(Student, Track, k, EP, n, m)$.

$\&transfer_IP(Student, Track, k, |EP|_{[n,m]})$ action designates an incoming transfer of the student *Student* to the EP interval $|EP|_{[n,m]}$ in slot k such that the student will graduate from this EP (while before the graduation the student can transfer from and to this EP several times). This action also indicates that the student has not studied at this EP earlier.

$Effects^+$: $graduate(Student, Track, EP)$.

$\&transfer_IRP(Student, Track, k, |EP|_{[n,m]})$ action designates an incoming transfer of the student *Student* to the EP interval $|EP|_{[n,m]}$ in slot k such that the student has studied at this EP earlier and will graduate from this EP in the future (while before the graduation the student can transfer from and to this EP several times).

$Effects^-$: $interrupted(Student, Track, k', EP, n', m')$.

$Effects^+$: $probation(Student, Track, k'+1, k-1, EP, m'+1, n-1)$.

$\&transfer_IT(Student, Track, k, |EP|_{[n,m]})$ action designates an incoming transfer of the student *Student* to the EP interval $|EP|_{[n,m]}$ in slot k such that the student will not graduate from this EP (before this action, the student may have studied at this EP before).

$Effects^+$: $not_graduate(Student, Track, EP)$.

$!transfer_OP(Student, Track, k, |EP|_{[n,m]})$ action designates an outgoing transfer of the student *Student* from the EP interval $|EP|_{[n,m]}$ in slot k such that the student will not graduate from this EP (after this action, the student can also carry out transfers to and from this EP).

$!transfer_OT(Student, Track, k, |EP|_{[n,m]})$ action designates an outgoing transfer of the student $Student$ to the EP interval $|EP|_{[n,m]}$ in slot k such that the student will return to this EP and will graduate from it in the future.

$Effects^{+}$: $interrupted(Student, Track, k, EP, n, m)$.

Decomposition methods for Basic track development and validation

One permanent transfer high-level scenario. This method splits *Degree* task into two sub-task. A student starts his (or her) education at one EP (designated using *Start_Degree* task, the EP interval is $|EP|_{[1,m]}$), then he (or she) transfers to another EP and graduates from it (designated using *Finish_Degree* task). $1 \leq F_1 < F_0$[4], $1 \leq m < EP_l$, where EP_l is the total number of semesters in EP.

$$Degree(S, Tr, Award^{Req}) \rightarrow$$
$$\langle Start_Degree(S, Tr(1, F_1), |EP|_{[1,m]}), Finish_Degree(S, Tr(F_1 + 1, F_0), Award^{Req})\rangle \quad \text{(B.1)}$$

Two permanent transfers high-level scenario. This method splits *Degree* task into three sub-tasks. Student starts his (or her) education at one EP (designated using *Start_Degree* task), then he (or she) transfers to another EP (designated using *Proceed_Degree* task) and then he (or she) transfers to the third degree, which he (or she) graduates from (designated using *Finish_Degree* task). $1 \leq F_1 < F_2 < F_0$, $1 \leq m < EP_l$.

$$Degree(S, Tr, Award^{Req}) \rightarrow$$
$$\langle Start_Degree(S, Tr(1, F_1), |EP|_{[1,m]}), Proceed_Degree(S, Tr(F_1 + 1, F_2), Award^{Req}),$$
$$Finish_Degree(S, Tr(F_2 + 1, F_0), Award^{Req})\rangle \quad \text{(B.2)}$$

Temporal transfer high-level scenario (probation period). This method splits *Degree* task into three sub-tasks. Student starts his (or her) education at one EP (designated using *Start_Degree_Probation* task), then he (or she) temporally transfers to another EP (designated using *Proceed_Degree* task) and then he (or she) returns to the first EP and graduates from it (designated using *Finish_Degree_fin* task). $1 \leq F_1 < F_2 < F_0$, $1 \leq m < n \leq EP_l$.

$$Degree(S, Tr, Award^{Req}) \rightarrow$$
$$\langle Start_Degree_Probation(S, Tr(1, F_1), |EP|_{[1,m]}), Proceed_Degree(S, Tr(F_1 + 1, F_2), Award^{Req}),$$
$$Finish_Degree_fin(S, Tr(F_2 + 1, F_0), |EP|_{[n,EP_l]})\rangle \quad \text{(B.3)}$$

[4] In this section, F_0 designates the number of the last slot in the track.

Temporal transfer scenario (probation period) within starting EP in high-level permanent transfer scenario. This method splits *Start_Degree* task into three sub-tasks. Student starts his (or her) education at one EP (designated using *Start_Degree_str* task), then he (or she) temporally transfers to another EP (designated using *Proceed_Degree* task) and then he (or she) returns to the first EP, transfers from it and never returns back (designated using *Start_Degree_fin* task). $1 < F_1 < F_2$, $1 \leq n < k \leq m$.

$$\begin{gathered} Start_Degree(S, Tr(1, F_2), |EP|_{[1,m]}) \rightarrow \\ \langle Start_Degree_str(S, Tr(1, F_1 - 1), |EP|_{[1,n]}), Proceed_Degree(S, Tr(F_1, F_2 - 1), Award^{Req}), \\ Start_Degree_fin(S, Tr(F_2, F_2), |EP|_{[k,m]})\rangle \end{gathered} \tag{B.4}$$

Temporal transfer scenario (probation period) within initial part of starting EP in high-level permanent transfer scenario. This method splits *Start_Degree_str* task into three sub-tasks in a recursive manner. Student starts his (or her) education at an EP that he (or she) will not graduate from (designated using *Start_Degree_str* task), then he (or she) temporally transfers to another EP (designated using *Proceed_Degree* task) and then he (or she) returns to the first EP and makes another temporal transfer (i.e., the student will return to this EP in the future) (designated using *Start_Degree_ret* task). $1 < F_1 < F_2$, $1 \leq n < k \leq m$.

$$\begin{gathered} Start_Degree_str(S, Tr(1, F_2), |EP|_{[1,m]}) \rightarrow \\ \langle Start_Degree_str(S, Tr(1, F_1 - 1), |EP|_{[1,n]}), Proceed_Degree(S, Tr(F_1, F_2 - 1), Award^{Req}), \\ Start_Degree_ret(S, Tr(F_2, F_2), |EP|_{[k,m]})\rangle \end{gathered} \tag{B.5}$$

Temporal transfer scenario (probation period) within final EP in high-level permanent transfer scenario. This method splits *Finish_Degree* task into three sub-tasks. After transferring to the EP that the student will graduate from (designated using *Finish_Degree_str* task), the student takes a probation period at another EP (designated using *Proceed_Degree* task). When it is finished, the student returns to the previous EP and graduates from it (designated using *Finish_Degree_fin* task). $S_0 < F_1 < F_0$, $n \leq m < k \leq EP_l$. Additionally, when it is considered that temporal transfer mobility scenarios used within the CEP are intersected, this task can be decomposed into task networks $\langle Proceed_Degree, Finish_Degree_fin\rangle$ and $\langle Finish_Degree_fin\rangle$.

$$\begin{gathered} Finish_Degree(S, Tr(S_0, F_0), Award^{Req}) \rightarrow \\ \langle Finish_Degree_str(S, Tr(S_0, S_0), |EP|_{[n,m]}), Proceed_Degree(S, Tr(S_0 + 1, F_1), Award^{Req}), \\ Finish_Degree_fin(S, Tr(F_1 + 1, F_0), |EP|_{[k,EP_l]})\rangle \end{gathered} \tag{B.6}$$

Temporal transfer scenario (probation period) within the last part of final EP in high-level permanent transfer scenario. This method splits $Finish_Degree_fin$ task into three sub-tasks in a recursive manner. After transferring to the EP that the student will graduate from and that he (or she) has already studied (designated using *Finish_Degree_ret* task), the student takes a probation period at another EP (designated using *Proceed_Degree* task). Then, the student returns to the EP, which he (or she) originally has transferred to (designated using $Finish_Degree_fin$ task), and graduates from it. $S_0 < F_1 < F_0$, $n \leq m < k \leq EP_l$.

$$\begin{gathered} Finish_Degree_fin(S, Tr(S_0, F_0), |EP|_{[n,EP_l]}) \rightarrow \\ \langle Finish_Degree_ret(S, Tr(S_0, S_0), |EP|_{[n,m]}), Proceed_Degree(S, Tr(S_0+1, F_1), Award^{Req}), \\ Finish_Degree_fin(S, Tr(F_1+1, F_0), |EP|_{[k,EP_l]}) \rangle \end{gathered} \tag{B.7}$$

Temporal transfer scenario (probation period) within initial part of higher-level probation period scenario. This method splits *Start_Degree_Probation* task into three sub-tasks in a recursive manner. Student starts his (or her) education at the EP (designated using *Start_Degree_Probation* task), then he (or she) temporally transfers to another EP (designated using *Proceed_Degree* task) and then he (or she) returns to the first EP and makes another transfer from it (but the student will return and graduate from this EP in the future) (designated using *Finish_Degree_ret* task). $1 < F_1 < F_0$, $1 \leq n < k \leq m$.

$$\begin{gathered} Start_Degree_Probation(S, Tr(1, F_0), |EP|_{[1,m]}) \rightarrow \\ \langle Start_Degree_Probation(S, Tr(1, F_1 - 1), |EP|_{[1,n]}), \\ Proceed_Degree(S, Tr(F_1, F_0 - 1), Award^{Req}), Finish_Degree_ret(S, Tr(F_0, F_0), |EP|_{[k,m]}) \rangle \end{gathered} \tag{B.8}$$

Permanent transfer scenario between 'intermediate' EPs. This method splits *Proceed_Degree* task into two sub-tasks in a recursive manner and introduces a permanent transfer between them. The student sequentially studies in two EPs. He (or she) has not started his (or her) higher education from any of them and will not graduate from any of them as well. $S_0 < F_0$, $1 < n \leq m < EP_l$. Additionally, when it is considered that temporal transfer mobility scenarios used within the CEP are intersected, this task can be decomposed into tasks $\langle Finish_Degree_str \rangle$ or $\langle Finish_Degree_ret \rangle$.

$$\begin{gathered} Proceed_Degree(S, Tr(S_0, F_0), Award^{Req}) \rightarrow \\ \langle Proceed_Degree(S, Tr(S_0, S_0), |EP|_{[n,m]}), Proceed_Degree(S, Tr(S_0+1, F_0), Award^{Req}) \rangle \end{gathered} \tag{B.9}$$

One-slot tasks processing: each of the following methods implements a one-slot task using

compound and primitive actions according to its definition.

$$Start_Degree(S, Tr(N^{Sl}, N^{Sl}), |EP|_{[n,m]}) \rightarrow \langle !admitT(S, Tr, N^{Sl}, |EP|_{[n,m]}), \\ \&choose_modules(S, Tr, N^{Sl}, |EP|_{[n,m]}), \&study_interval(S, Tr, N^{Sl}, |EP|_{[n,m]}), \\ !transfer_OP(S, Tr, N^{Sl}, |EP|_{[n,m]})\rangle \tag{B.10}$$

$$Finish_Degree(S, Tr(N^{Sl}, N^{Sl}), Award^{Req}) \rightarrow \langle \&transfer_IP(S, Tr, N^{Sl}, |EP|_{[n,m]}), \\ \&choose_modules(S, Tr, N^{Sl}, |EP|_{[n,m]}), \&study_interval(S, Tr, N^{Sl}, |EP|_{[n,m]}), \\ !graduate(S, Tr, N^{Sl}, |EP|_{[n,m]})\rangle \tag{B.11}$$

$$Proceed_Degree(S, Tr(N^{Sl}, N^{Sl}), [|EP|_{[n,m]} \ or \ Award^{Req}]) \rightarrow \\ \langle \&transfer_IT(S, Tr, N^{Sl}, |EP|_{[n,m]}), \&choose_modules(S, Tr, N^{Sl}, |EP|_{[n,m]}), \\ \&study_interval(S, Tr, N^{Sl}, |EP|_{[n,m]}), !transfer_OP(S, Tr, N^{Sl}, |EP|_{[n,m]})\rangle \tag{B.12}$$

$$Start_Degree_Probation(S, Tr(N^{Sl}, N^{Sl}), |EP|_{[n,m]}) \rightarrow \langle !admitP(S, Tr, N^{Sl}, |EP|_{[n,m]}), \\ \&choose_modules(S, Tr, N^{Sl}, |EP|_{[n,m]}), \&study_interval(S, Tr, N^{Sl}, |EP|_{[n,m]}), \\ !transfer_OT(S, Tr, N^{Sl}, |EP|_{[n,m]})\rangle \tag{B.13}$$

$$Start_Degree_str(S, Tr(N^{Sl}, N^{Sl}), |EP|_{[n,m]}) \rightarrow \langle !admitT(S, Tr, N^{Sl}, |EP|_{[n,m]}), \\ \&choose_modules(S, Tr, N^{Sl}, |EP|_{[n,m]}), \&study_interval(S, Tr, N^{Sl}, |EP|_{[n,m]}), \\ !transfer_OP(S, Tr, N^{Sl}, |EP|_{[n,m]})\rangle \tag{B.14}$$

$$Start_Degree_fin(S, Tr(N^{Sl}, N^{Sl}), |EP|_{[n,m]}) \rightarrow \langle \&transfer_IT(S, Tr, N^{Sl}, |EP|_{[n,m]}), \\ \&choose_modules(S, Tr, N^{Sl}, |EP|_{[n,m]}), \&study_interval(S, Tr, N^{Sl}, |EP|_{[n,m]}), \\ \&transfer_OP(S, Tr, N^{Sl}, |EP|_{[n,m]})\rangle \tag{B.15}$$

$$Start_Degree_ret(S, Tr(N^{Sl}, N^{Sl}), |EP|_{[n,m]}) \rightarrow \langle \&transfer_IT(S, Tr, N^{Sl}, |EP|_{[n,m]}), \\ \&choose_modules(S, Tr, N^{Sl}, |EP|_{[n,m]}), \&study_interval(S, Tr, N^{Sl}, |EP|_{[n,m]}), \\ !transfer_OP(S, Tr, N^{Sl}, |EP|_{[n,m]})\rangle \tag{B.16}$$

$$Finish_Degree_str(S, Tr(N^{Sl}, N^{Sl}), |EP|_{[n,m]}) \rightarrow \langle \&transfer_IP(S, Tr, N^{Sl}, |EP|_{[n,m]}), \&choose_modules(S, Tr, N^{Sl}, |EP|_{[n,m]}), \&study_interval(S, Tr, N^{Sl}, |EP|_{[n,m]}), !transfer_OT(S, Tr, N^{Sl}, |EP|_{[n,m]})\rangle \quad (B.17)$$

$$Finish_Degree_ret(S, Tr(N^{Sl}, N^{Sl}), |EP|_{[n,m]}) \rightarrow \langle \&transfer_IRP(S, Tr, N^{Sl}, |EP|_{[n,m]}), \&choose_modules(S, Tr, N^{Sl}, |EP|_{[n,m]}), \&study_interval(S, Tr, N^{Sl}, |EP|_{[n,m]}), !transfer_OT(S, Tr, N^{Sl}, |EP|_{[n,m]})\rangle \quad (B.18)$$

$$Finish_Degree_fin(S, Tr(N^{Sl}, N^{Sl}), |EP|_{[n,m]}) \rightarrow \langle \&transfer_IRP(S, Tr, N^{Sl}, |EP|_{[n,m]}), \&choose_modules(S, Tr, N^{Sl}, |EP|_{[n,m]}), \&study_interval(S, Tr, N^{Sl}, |EP|_{[n,m]}), !graduate(S, Tr, N^{Sl}, |EP|_{[n,m]})\rangle \quad (B.19)$$

Planning tasks definitions for obligatory low-level processes

$Choose_modules_find_groups(Student, Track, k, |EP|_{[n,m]})$ compound task is used to close all groups within the EP interval $|EP|_{[n,m]}$ using recursion. When backtrack is carried out, different modules selection variants for the groups closure are tried.

$Choose_modules_group(Student, Track, k, Group, |EP|_{[n,m]})$ compound task is used to process the group $Group$ and recursively try all possible variants of the modules selection searching for the variants when the group can be closed.

$!choose_module(Student, Track, k, Mod, Group, |EP|_{[n,m]}, InEPint)$ action is carried out when the optional module Mod within the group $Group$ is selected. $InEPint$ parameter is used to indicate if this group is contained in the EP interval that will be studied by the student or only recognised for him (or her).

$Effects^{+}$: $module_chosen(Student, Track, k, Mod, Group, EP, n, m, InEPint)$.

$!close_group(Student, Track, k, Group, |EP|_{[n,m]}, InEPint)$ action is used when the group of optional modules $Group$ within the EP interval $|EP|_{[n,m]}$ is closed.

$Effects^{+}$: $group_is_closed(Student, Track, k, Group, EP, n, m, InEPint)$.

$Find_sems(Student, |EP|_{[n,m]}, i, Track, k)$ compound task is used to recursively process all modules within the EP interval $|EP|_{[n,m]}$. Parameter $n+i$ is used to determine the next semester.

$\&study_sem(Student, |EP|_{[n,m]}, Sem, Track, k)$ compound action indicates that the student $Student$ has studied semester Sem in slot k of the track $Track$.

$Find_modules(Student, |EP|_{[n,m]}, Sem, Track, k)$ compound task is used to find and process all modules within the semester Sem.

$!stud_mod(Student, |EP|_{[n,m]}, Sem, Mod, Track, k)$ action designates that the student $Student$ has studied module Mod in slot k of the track $Track$.

$Effects^{+}$: $studied_mod(Student, Track, k, Mod, EP, n, m)$.

Decomposition methods for obligatory low-level processes

Compound action $\&choose_modules$ **decomposition method**. Using this method, the recursive task $Choose_modules_find_groups$ is generated and it is designated that it can contain 'pseudo-parallel' actions.

$$\begin{gathered} \&choose_modules(S, Tr, k, |EP|_{[n,m]}) \rightarrow \\ \langle !CA_start(choose_modules(S, Tr, k, |EP|_{[n,m]}), \emptyset, \emptyset, \emptyset), !concur_start(), \\ Choose_modules_find_groups(S, Tr, k, |EP|_{[n,m]}), \\ !concur_end(), !CA_end(choose_modules(S, Tr, k, |EP|_{[n,m]}), \emptyset, \emptyset, \emptyset)\rangle \end{gathered} \tag{B.20}$$

Recursive groups processing method. This method contains two branches that are tried sequentially. In the first branch, one group that has not been processed yet is marked using a flag and processed. If all groups are marked, the second branch is carried out and the task is decomposed into an empty task network. This method was designed such that during the backtrack different orderings for groups are not tried.

$$\begin{gathered} Choose_modules_find_groups(S, Tr, k, |EP|_{[n,m]}) \rightarrow \\ \langle Choose_modules_group(S, Tr, k, Group, |EP|_{[n,m]}), !change_line(), \\ Choose_modules_find_groups(S, Tr, k, |EP|_{[n,m]})\rangle \mid \langle \emptyset \rangle \end{gathered} \tag{B.21}$$

Recursive modules selection methods. This method contains two alternative branches for decomposition of $Choose_modules_group$ task referring to a group of optional modules. The first branch straight away tries to close the group. If this fails, in the second branch one module is selected and the task $Choose_modules_group$ is carried out again for this task. When this method backtracks, different variants for the modules selection are tried.

$$\begin{gathered} Choose_modules_group(S, Tr, k, Group, |EP|_{[n,m]}) \rightarrow \\ \langle !close_group(S, Tr, k, Group, |EP|_{[n,m]}, 1)\rangle \mid \langle !choose_module(S, Tr, k, Mod, Group, |EP|_{[n,m]}, 1), \\ Choose_modules_group(S, Tr, k, Group, |EP|_{[n,m]})\rangle \end{gathered} \tag{B.22}$$

Compound action $\&study_interval$ **decomposition method**. Using this method, the re-

cursive task $Find_sems$ is generated.

$$\&study_interval(S, Tr, k, |EP|_{[n,m]}) \rightarrow$$
$$\langle !CA_start(\&study_interval(S, Tr, k, |EP|_{[n,m]}), \emptyset, \emptyset, \emptyset), Find_sems(S, |EP|_{[n,m],0,Tr,k}),$$
$$!CA_end(\&study_interval(S, Tr, k, |EP|_{[n,m]}), \emptyset, \emptyset, \emptyset)\rangle \tag{B.23}$$

Recursive semesters processing method. This method contains two branches that are tried sequentially. In the first branch, the next unprocessed semester is processed. If all semesters within the EP interval have been processed, the second branch is carried out and the recursion is stopped.

$$Find_sems(S, |EP|_{[n,m],i,Tr,k}) \rightarrow$$
$$\langle \&study_sem(S, |EP|_{[n,m]}, Sem_{n+i}, Tr, k), Find_sems(S, |EP|_{[n,m]}, i+1, Tr, k)\rangle \mid \langle\emptyset\rangle \tag{B.24}$$

Compound action $\&study_sem$ **decomposition method**. Using this method, the recursive task $Find_modules$ is generated and it is designated that it can contain 'pseudo-parallel' actions.

$$\&study_sem(S, |EP|_{[n,m],Sem,Tr,k}) \rightarrow$$
$$\langle !CA_start(study_sem(S, |EP|_{[n,m]}, Sem, Tr, k), \emptyset, \emptyset, \emptyset), !concur_start(),$$
$$Find_modules(S, |EP|_{[n,m]}, Sem, Tr, k),$$
$$!concur_end(), !CA_end(study_sem(S, |EP|_{[n,m]}, Sem, Tr, k), \emptyset, \emptyset, \emptyset)\rangle \tag{B.25}$$

Recursive modules processing method. This method contains two branches that are tried sequentially. In the first branch, the one unprocessed module is marked using a flag and processed. If all modules within the EP interval has been processed, the second branch is carried out and the recursion is stopped. This method was designed such that during the backtrack it does not try different module orderings.

$$Find_modules(S, |EP|_{[n,m]}, Sem, Tr, k) \rightarrow$$
$$\langle !study_mod(S, |EP|_{[n,m]}, Sem, Mod, Tr, k), !change_line(),$$
$$Find_modules(S, |EP|_{[n,m]}, Sem, Tr, k)\rangle \mid \langle\emptyset\rangle \tag{B.26}$$

Appendix C

ANTLR grammar for XPath parsing and AST generation

grammar *XPathGr*;

options {
output = 'AST'; }

tokens {
RELPATH; **ABSATH**; **STEP**; **GOUP**; **UNION**; **OR**; **AND**; **EQUAL**; **REL**; **ADD**; **MULT**; **ANY**; **SELF**; }

PATHSEP : '/'; **LPAR** : '('; **RPAR** : ')'; **LBRAC** : '['; **RBRAC** : ']'; **MINUS** : '-'; **PLUS** : '+'; **DOT** : '.'; **ASTER** : '*'; **DOTDOT** : '..'; **AT** : '@'; **COMMA** : ','; **PIPE** : '|'; **LESS** : '<'; **MORE** : '>'; **LE** : '<='; **GE** : '>='; **EQ** : '='; **NEQ** : '!='; **COLON** : ':';

main : expr;

locationPath :
relativeLocationPath ->^(**RELPATH** relativeLocationPath)
|absoluteLocationPathNoroot->^(**ABSATH** absoluteLocationPathNoroot) ;

absoluteLocationPathNoroot : **PATHSEP**! relativeLocationPath;

relativeLocationPath : step (**PATHSEP**! step)*;

step :
nodeTest predicate* ->^(**STEP** nodeTest predicate*)
| abbreviatedStep ->^(**STEP** abbreviatedStep)
| **AT** nodeTest predicate* ->^(**STEP** predicate*);

nodeTest :
nameTest
| **NodeTypeNode LPAR RPAR** ->^(**ANY**)
| **NodeTypeOthers**! **LPAR**! **RPAR**!
| 'processing-instruction'! **LPAR**! **Literal**! **RPAR**!;

predicate : **LBRAC**! expr **RBRAC**!;

abbreviatedStep :
DOT ->^(**SELF**)
| **DOTDOT** ->^(**GOUP**);

expr : orExpr;

primaryExpr :
variableReference!
| **LPAR**! expr **RPAR**!

```
  | Literal!
  | Number!
  | functionCall!;

functionCall :   functionName LPAR ( expr ( COMMA expr )* )? RPAR;

unionExprNoRoot :   pathExprNoRoot (PIPE unionExprNoRoot)?
->^(UNION pathExprNoRoot unionExprNoRoot?)
  | PATHSEP PIPE unionExprNoRoot ->^(UNION unionExprNoRoot);

pathExprNoRoot :
  locationPath
  | filterExpr ( PATHSEP! relativeLocationPath)? ;

filterExpr :   primaryExpr predicate* ;

orExpr :   E = andExpr ('or' F = andExpr)* ->^(OR $E $F*);

andExpr :   G = equalityExpr ('and' H = equalityExpr)* ->^ (AND $G $H*);

equalityExpr :
  A = relationalExpr ((EQ|NEQ) B = relationalExpr)*   ->^ (EQUAL $A $B*);

relationalExpr :
  C = additiveExpr ((LESS|MORE|LE|GE) D = additiveExpr)*   ->^ (REL $C $D*);

additiveExpr :
  I = multiplicativeExpr ((MINUS|PLUS) J = multiplicativeExpr)*
->^ (ADD $I $J*);

multiplicativeExpr :
  K = unaryExprNoRoot ((MUL|'div'|'mod') L = multiplicativeExpr)?
->^ (MULT $K $L?)
  | PATHSEP (('div'|'mod') multiplicativeExpr)?
  ->^(MULT multiplicativeExpr?);

unaryExprNoRoot :   MINUS* unionExprNoRoot;

qName :   NCName (COLON! NCName)?;

functionName :   qName;

variableReference :   '$' qName;

nameTest :
   ASTER ->^(ANY)
  | NCName COLON! ASTER ->NCNameANY
  | qName;

NodeTypeOthers :
  'comment'
  | 'text'
  | 'processing-instruction';

NodeTypeNode :   'node';

Number :   Digits ('.' Digits? )?
  | '.' Digits;

fragment
Digits : ('0'..'9')+;

Literal :
   '  "  '  ~'  "  '*  '  "  '
```

| ’ ′ ’ ’ ″ ’* ’ ″ ’;

Whitespace : (’ ’ | ’\t’ | ’\n’ | ’\r’)$^+$ {$channel = HIDDEN;} ;

NCName : **NCNameStartChar NCNameChar*** ;

`fragment`
NCNameStartChar : ’A’..’Z’ | ’_’ | ’a’..’z’ | ’\u00C0’ .. ’\u00D6’
| ’\u00D8’ .. ’\u00F6’ | ’\u00F8’ .. ’\u02FF’ | ’\u0370’ .. ’\u037D’
| ’\u037F’ .. ’\u1FFF’ | ’\u200C’ .. ’\u200D’ | ’\u2070’ .. ’\u218F’
| ’\u2C00’ .. ’\u2FEF’ | ’\u3001’ .. ’\uD7FF’ | ’\uF900’ .. ’\uFDCF’
| ’\uFDF0’ .. ’\uFFFD’ ;

`fragment`
NCNameChar :
NCNameStartChar | ’-’ | ’.’ | ’0’ .. ’9’
| ’\u00B7’ | ’\u0300’ .. ’\u036F’ | ’\u203F’ .. ’\u2040’ ;

Appendix D

Case study 2 planning environment specifications

Educational programmes specification

EP1: MSc in Advanced Computer Science. University of Birmingham.[1]

Award	Level	MSc (UK NQF)
	Title	Advanced Computer Science
	Area	Computing_48 (ISCED)
	Education provider	School of Computer Science, University of Birmingham

Structure						
ID	Name	Cr.	Sem. 1 (09 - 12)[2]	Sem. 2 (01 - 04)	Sem. 3 (05 - 09)	Prereq
B_1	Research Skills	10	Oblig.			
B_2	Compilers and Languages	10		Opt.($Group_2$)		
B_3	Human Computer Interaction	10	Opt.($Group_1$)			
B_4	Distributed Systems	10	Opt.($Group_1$)			
B_5	Parallel Programming	10		Opt.($Group_2$)		
B_6	Enterprise Systems	20		Opt.($Group_2$)		B_{13}, B_{14}
B_7	Formal Methods	10		Opt.($Group_2$)		B_{15}, B_{18}
B_8	Second semester mini-project	30		Oblig.		B_{14}, B_{17}, B_{18}
B_9	Project	60			Oblig.	
B_{10}	Networks	10	Opt.($Group_1$)			
B_{11}	Intelligent Robotics	20	Opt.($Group_1$)			
B_{12}	Computer Security	20	Opt.($Group_1$)			

[1] This EP is a part of referred MSc. It contains only a subset of optional modules, contained in the original MSc. Additionally, minor modifications were introduced into MSc modules, i.e. prerequisites were added for modules B_7 and B_8, start and end dates of semesters were adapted.

[2]Here and in subsequent tables numbers in brackets in columns referring to EP semesters are sequence numbers (in a year) of the start and end months for these semesters.

- 50 Credits should be chosen from $Group_1$
- 30 Credits should be chosen from $Group_2$

Credit values for modules in EP_1 (the third column in the EP structure table) are specified in educational credit units adopted in UK. We assume that one such credit unit is equal to 10 notional hours.

EP2: MSc in Robotics. University of Birmingham.[3]

Award	Level	MSc (UK NQF)
	Title	Robotics
	Area	Computing_48 (ISCED)
	Education provider	School of Computer Science, University of Birmingham

Structure						
ID	Name	Cr.	Sem. 1 (09 - 12)	Sem. 2 (01 - 04)	Sem. 3 (05 - 09)	Prereq
C_1	Intelligent Robotics	20	Oblig.			
C_2	Robot Vision	20	Oblig.			
C_3	Advanced Robotics	20		Oblig.		
C_4	Graphics	10		Opt.($Group_4$)		
C_5	Planning	10		Opt.($Group_4$)		
C_6	Machine Learning	10	Opt.($Group_3$)			
C_7	Int.to Evolutionary Computation	10	Opt.($Group_3$)			
C_8	Int.to Neural Computation	10	Opt.($Group_3$)			
C_9	Intelligent Data Analysis	10		Opt.($Group_4$)		
C_{10}	Computational Vision	10		Opt.($Group_4$)		C_{13}
C_{11}	Computational Modelling with MATLAB	10		Opt.($Group_4$)		
C_{12}	Component-based Software	10	Opt.($Group_3$)			
C_{13}	Project	60			Oblig.	

- 20 Credits should be chosen from $Group_3$
- 40 Credits should be chosen from $Group_4$

[3]This EP is a part of referred MSc. It contains only a subset of optional modules, contained in the original MSc. Minor modifications were introduced into the MSc modules, start and end dates of semesters were adapted.

Credit values for modules in EP_2 (the third column in the EP structure table) are specified in educational credit units adopted in UK. We assume that one such credit unit is equal to 10 notional hours.

Additional modules for prerequisites		
ID	Name	Cr.
B_{13}	Databases	15
B_{14}	Object-Oriented Programming	10
B_{15}	Discrete mathematics	10
B_{16}	Object-Oriented Design	20
B_{17}	Software Engineering Processes	10
C_{13}	Analytical Geometry	10

EP3: MSc in Software Engineering. BMSTU. [4]

Award	Level	MSc (RF NQF)
	Title	Software Engineering
	Area	Computing_48 (ISCED)
	Education provider	Bauman Moscow State Technical University (BMSTU)

[4]Minor modifications were introduced into the MSc modules, i.e. prerequisites were added for modules A_3 and A_6, start and end dates of semesters were adapted.

Structure							
ID	Name	Cr.	Sem. 1 (09 - 12)	Sem. 2 (01 - 08)	Sem. 3 (09 - 12)	Sem. 4 (01 - 06)	Prereq
A_1	Distributed Systems	4	Oblig.				
A_2	Research Work	4	Oblig.				
A_3	Compilers Development	4	Oblig.				A_{22}
A_4	Parallel Computations	3	Oblig.				
A_5	Real Time Systems	4	Oblig.				
A_6	Object Oriented Analysis and Design	3	Oblig.				A_{23}
A_7	Network Protocols Development	5	Opt. ($Group_5$)				A_{24}
A_8	Research Methods	3	Oblig.				
A_9	Philosophy	2		Oblig.			
A_{10}	Digital Signals Processing	6		Oblig.			
A_{11}	Intellectual Data Analysis	5	Opt. ($Group_5$)				
A_{12}	High-Performance Computing	4		Oblig.			
A_{13}	Research Work	9		Oblig.			
A_{14}	Robotics and Automatic Systems	4		Oblig.			
A_{15}	Software Development Methodology	5		Opt. ($Group_6$)			
A_{16}	Decision Theory	5		Opt. ($Group_6$)			
A_{17}	Philosophy (cont.)	2			Oblig.		
A_{18}	Foreign Language	3			Oblig.		
A_{19}	Operational Systems Design	6			Opt. ($Group_7$)		
A_{20}	Software Verification	4			Oblig.		
A_{21}	Research Work	8			Oblig.		
A_{22}	Pattern Recognition	7			Oblig.		
	Algorithms						
A_{23}	Computational Linguistic	6			Opt. ($Group_7$)		
A_{23}	Project	30				Oblig.	

- One and only module should be chosen from $Group_5$
- One and only module should be chosen from $Group_6$
- One and only module should be chosen from $Group_7$

Credit values for modules in EP_3 (the third column in the EP structure table) are specified in Russian Federation educational credit units. One credit unit is equal to 36 notional hours.

Additional modules for prerequisites		
ID	Name	Cr.
A_{22}	Discrete Mathematics	4
A_{23}	Object-Oriented Programming	5
A_{24}	Formal Languages Theory	3

Policy specification

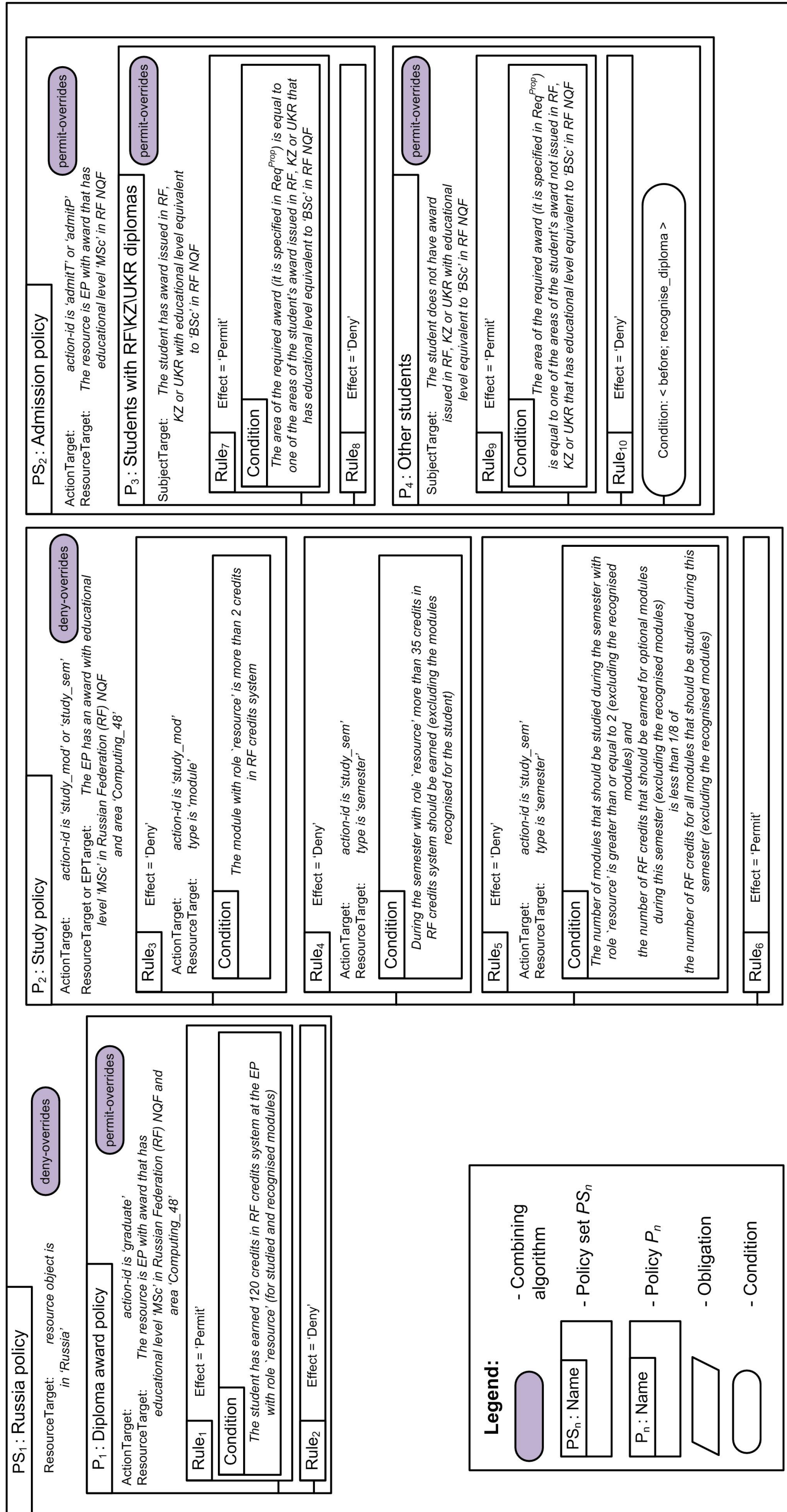

Figure D.1: Schema for *Russia* policy set

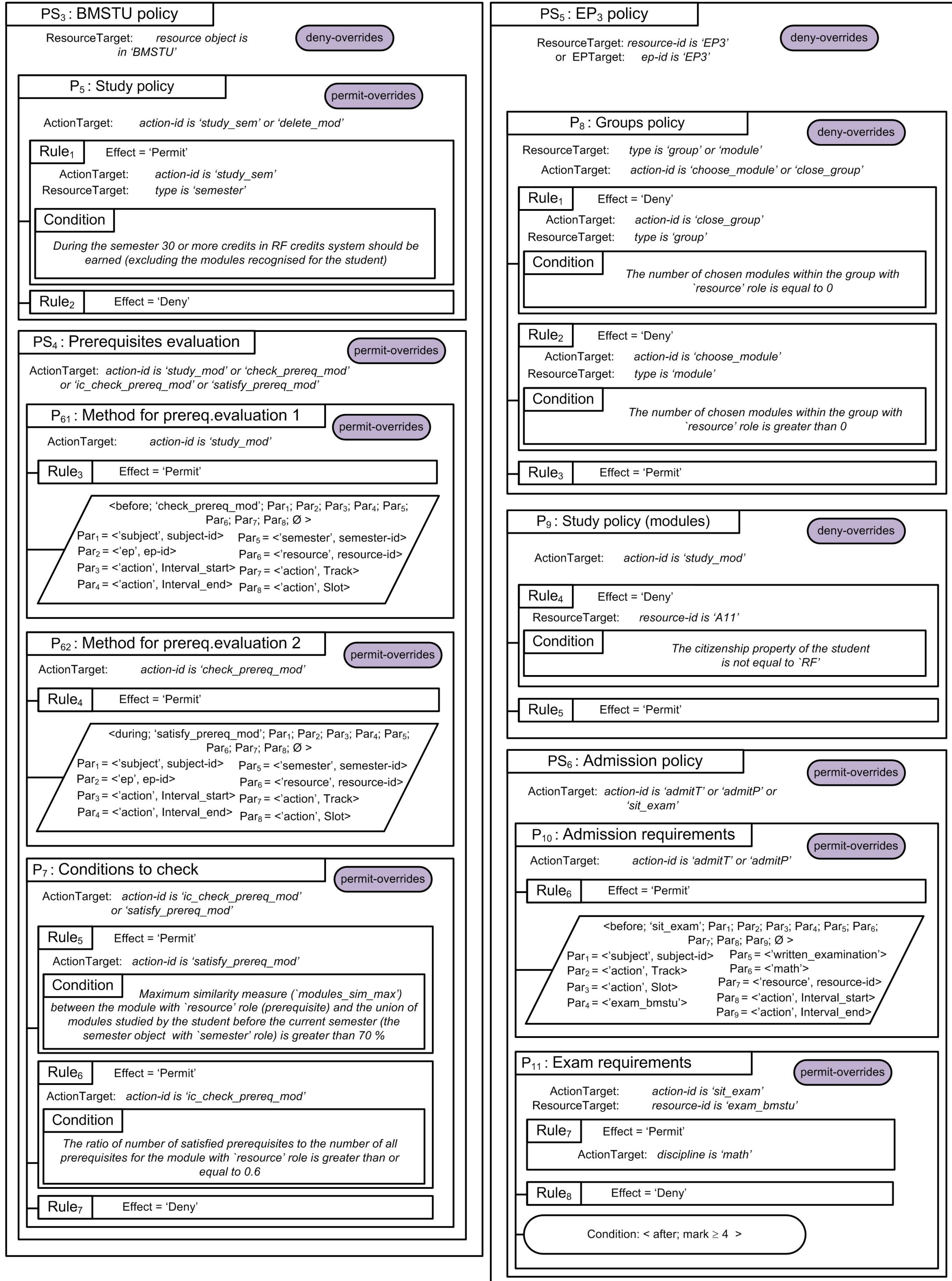

Figure D.2: Schemas for $BMSTU$ and EP_3 policy sets

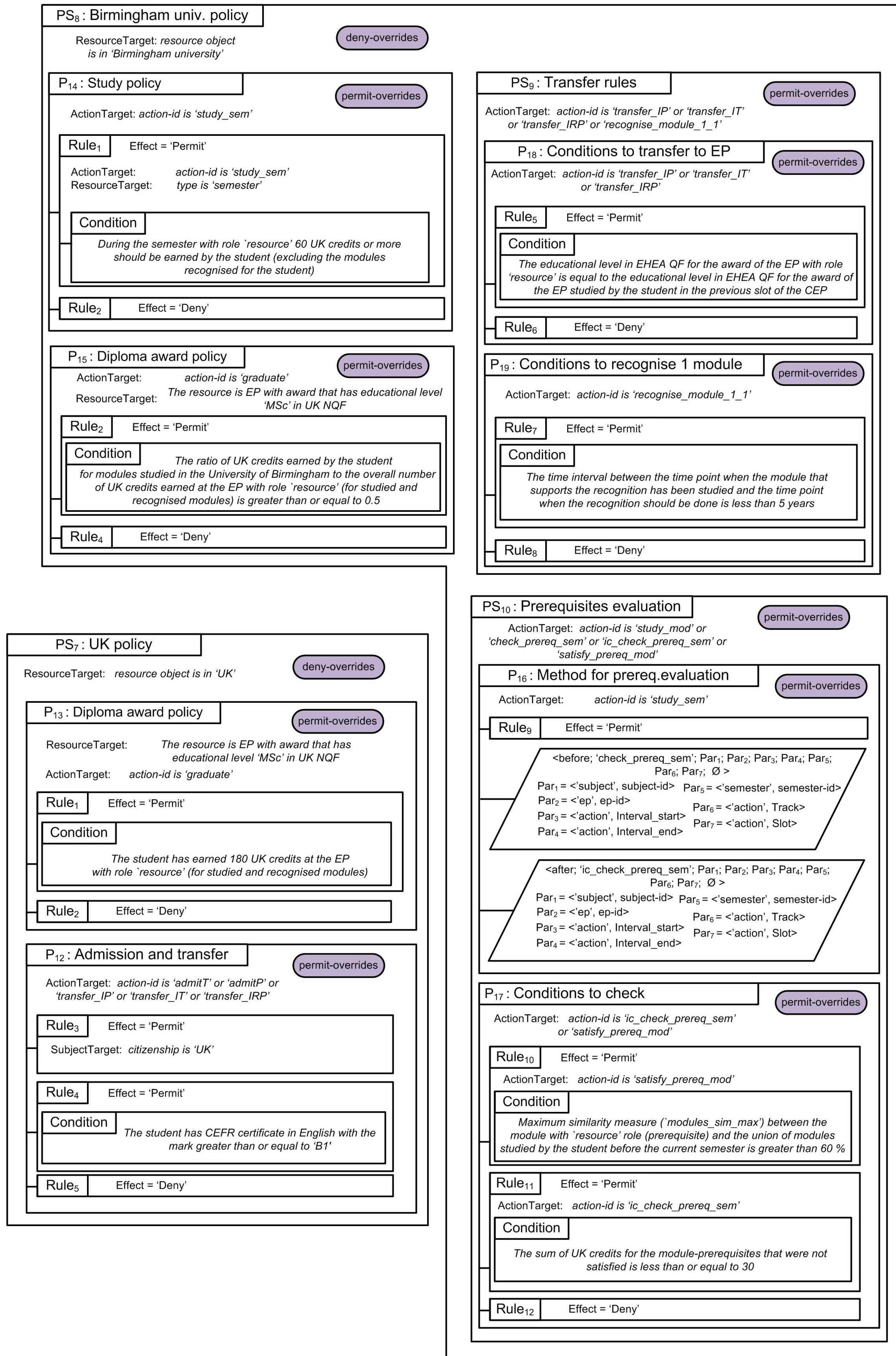

Figure D.3: Schemas for *UK* and *University of Birmingham* policy sets

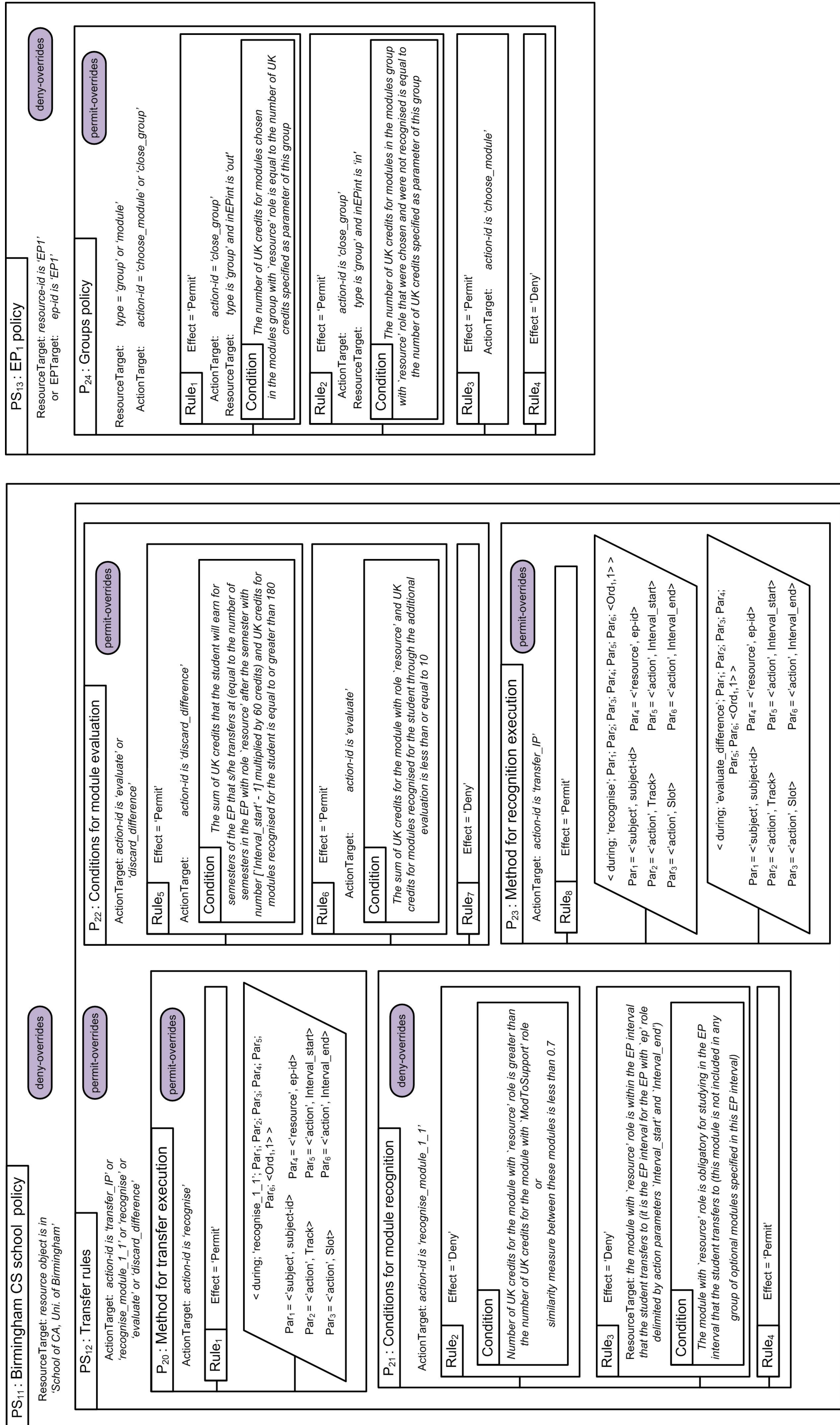

Figure D.4: Schemas for *School of Computer Science, University of Birmingham* and Msc in Advanced Computer Science EP_1 policy sets

Appendix E

Experimental results for performance analysis

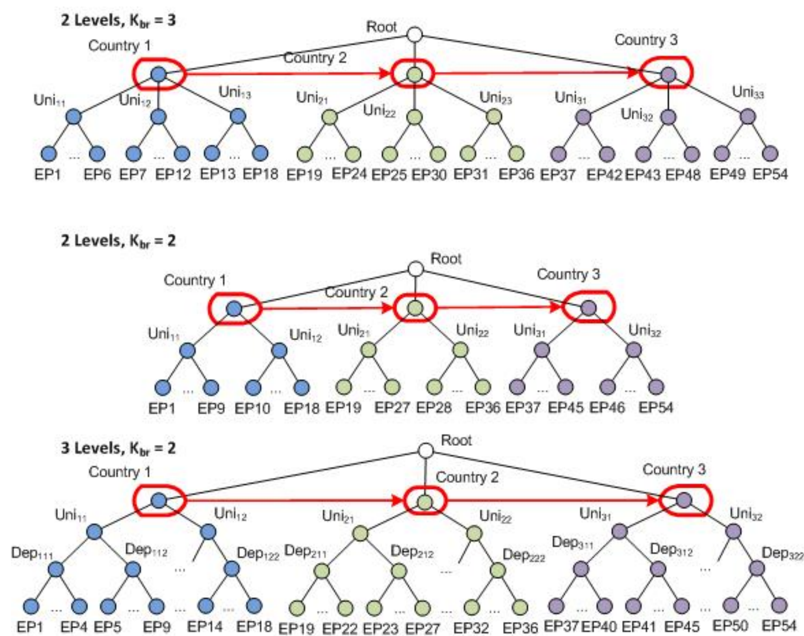

Figure E.1: Examples of domains trees used for performance analysis

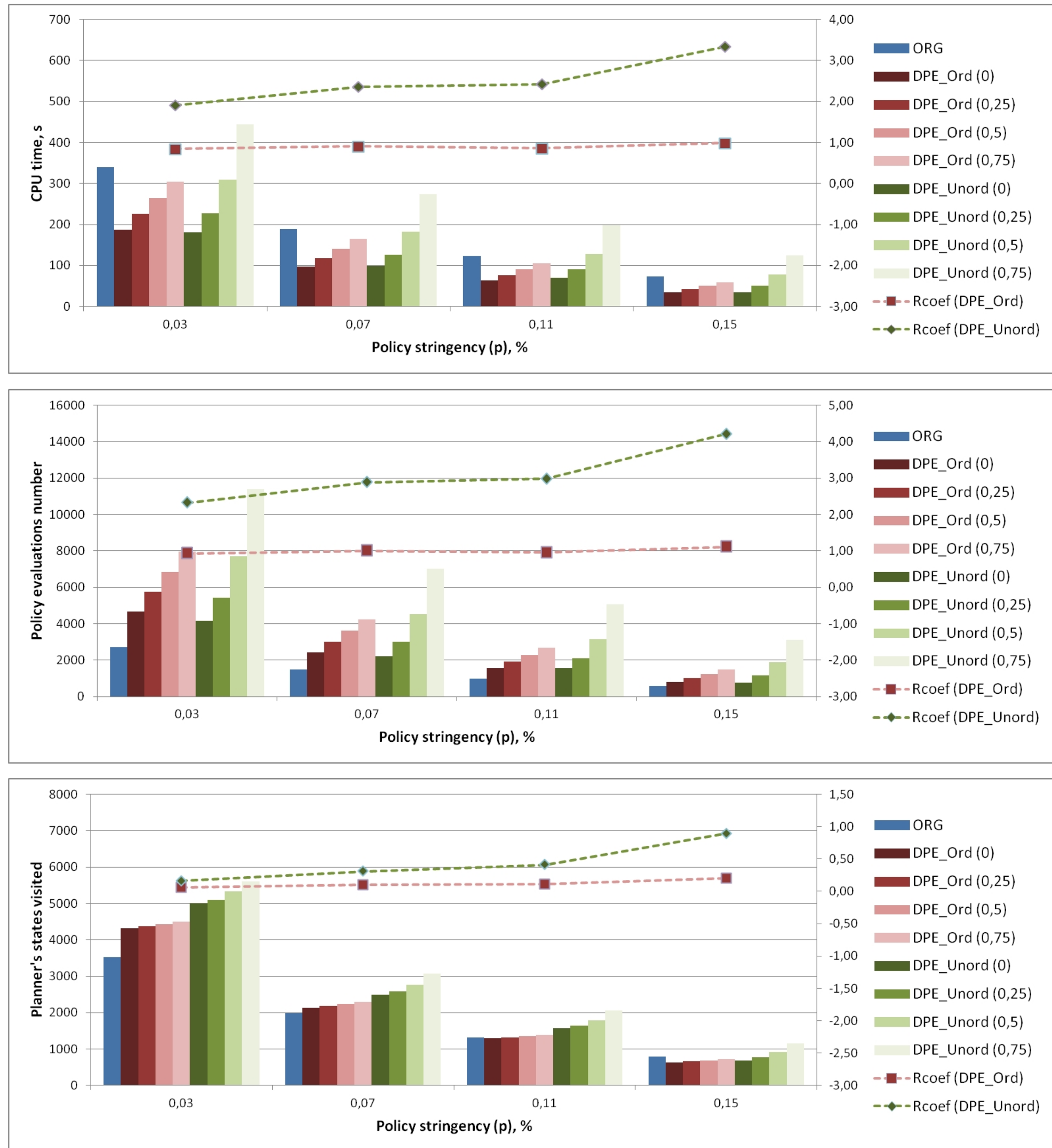

Figure E.2: Experimental results for different values of z (Experiment 1.2)

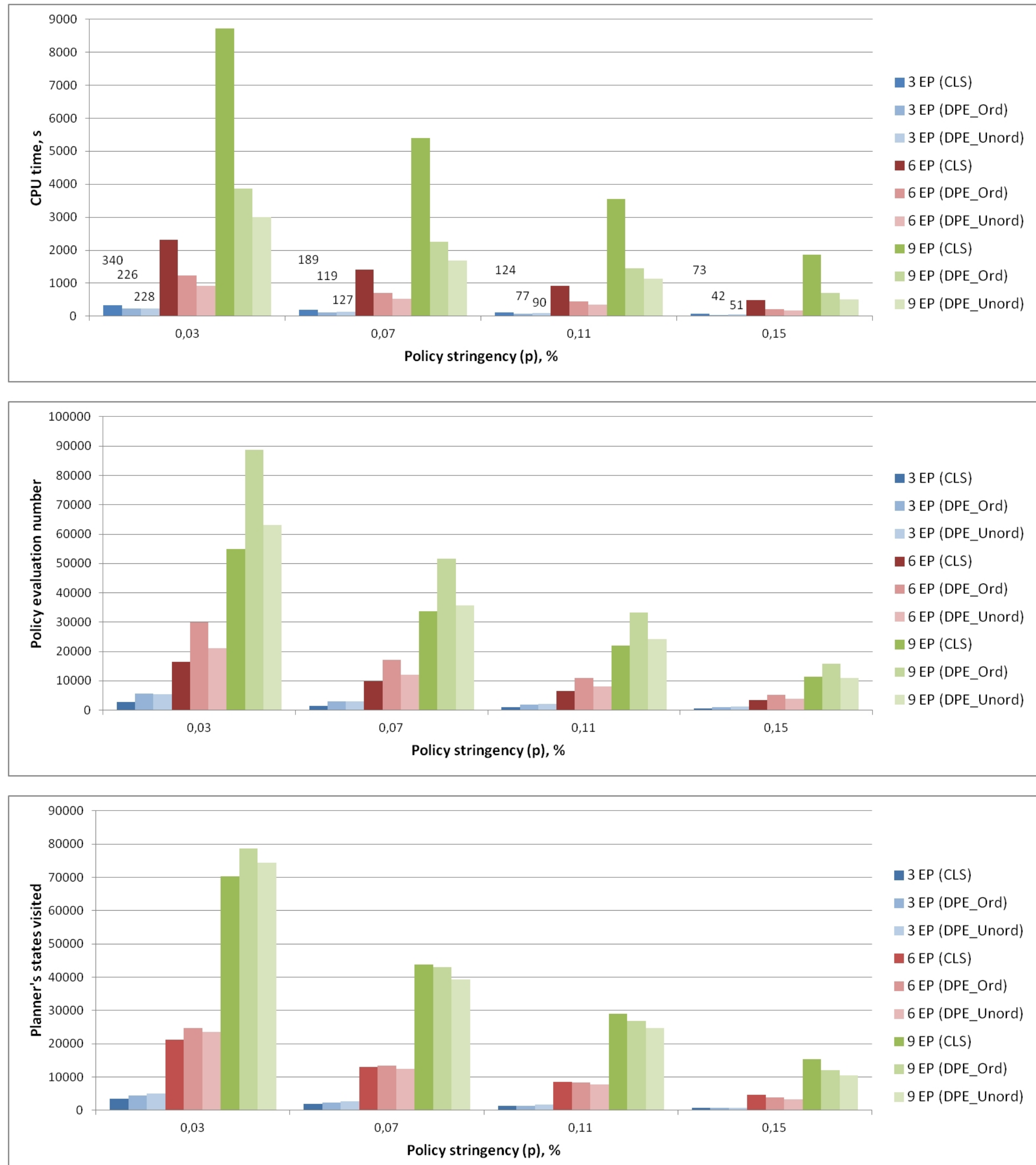

Figure E.3: Experimental results for different number of EPs (Experiment 2.1)