

# Open Research Online

---

The Open University's repository of research publications  
and other research outputs

## A Generic Library of Problem Solving Methods for Scheduling Applications

### Thesis

#### How to cite:

Rajpathak, Dnyanesh (2005). A Generic Library of Problem Solving Methods for Scheduling Applications. PhD thesis. The Open University.

For guidance on citations see [FAQs](#).

© 2005 Dnyanesh Rajpathak

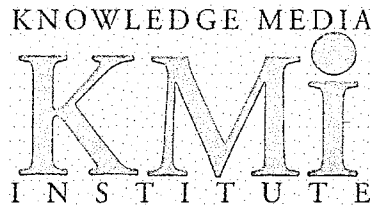
Version: Version of Record

---

Copyright and Moral Rights for the articles on this site are retained by the individual authors and/or other copyright owners. For more information on Open Research Online's data [policy](#) on reuse of materials please consult the policies page.

---

[oro.open.ac.uk](http://oro.open.ac.uk)

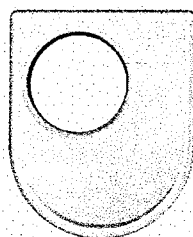


# **A Generic Library of Problem Solving Methods for Scheduling Applications**

Dnyanesh Rajpathak BEng. MSc.

Thesis submitted in partial fulfilment of the requirements for  
the degree of Doctor of Philosophy  
In Artificial Intelligence

Submitted - 30<sup>th</sup> September, 2004



The Open University

DATE OF SUBMISSION 27 SEPTEMBER 2004  
DATE OF AWARD 27 APRIL 2005

ProQuest Number: C821664

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



ProQuest C821664

Published by ProQuest LLC (2019). Copyright of the Dissertation is held by the Author.

All rights reserved.

This work is protected against unauthorized copying under Title 17, United States Code  
Microform Edition © ProQuest LLC.

ProQuest LLC.  
789 East Eisenhower Parkway  
P.O. Box 1346  
Ann Arbor, MI 48106 – 1346

# Abstract

In this thesis we propose a generic library of scheduling problem-solving methods. As a first approximation, scheduling can be defined as an assignment of jobs and activities to resources and time ranges in accordance with a number of constraints and requirements. In some cases optimisation criteria may also be included in the problem specification.

Although, several attempts have been made in the past at developing the libraries of scheduling problem-solvers, these only provide limited coverage. Many lack generality, as they subscribe to a particular scheduling domain. Others simply implement a particular problem-solving technique, which may be applicable only to a subset of the space of scheduling problems. In addition, most of these libraries fail to provide the required degree of depth and precision, which is needed both to obtain a formal account of scheduling problem solving and to provide effective support for development of scheduling applications by reuse.

Our library subscribes to the Task-Method-Domain-Application (TMDA) knowledge modelling framework, which provides a structured organisation for the different components of the library. In line with the organisation proposed by TMDA, we first developed a generic scheduling task ontology, which formalises the space of scheduling problems independently of any particular application domain, or problem solving method. Then we constructed a task-specific, but domain independent model of scheduling problem-solving, which generalises from the variety of approaches to scheduling problem-solving, which can be found in literature. The generic nature of this model was demonstrated by constructing seven methods for scheduling, as alternative specialisation of the model. Finally, we validated our library on a number of applications to demonstrate its generic nature and effective support for the analysis and development of scheduling applications.

To the memories of my father...  
My family and Dr. Ashok Marathe,  
with love and gratitude

# Acknowledgements

The writing of a dissertation can be a lonely and isolating experience, yet it is obviously not possible without the personal and practical support of the numerous people. Thus, I am greatly indebted to my supervisors, Prof. Enrico Motta, Dr. Zdenek Zdrahal, and Dr. Rajkumar Roy, for their advice and support throughout this work.

I would like to express my sincere thanks to Prof. Motta, who helped me at different ‘levels’. At an abstract level, Prof. Motta has been an excellent supervisor providing insightful comments and constructive criticism throughout this research work. At a theoretical level, all the discussions and intellectually stimulating ‘question answering’ that went through the PhD supervision meetings with Prof. Motta not only helped me in clarifying my understanding of the fine-grained details of knowledge modelling, but also helped me in formalising the scientific issues involved in scheduling in a more structured and coherent way. At a personal level, all the support and encouragement that Prof. Motta provided while completing this work is difficult to express in words. Moreover, I would like to thank Prof. Motta for thoroughly reading and providing fine-grained comments on several drafts of this thesis. I would also like to thank my other two supervisors Dr. Zdrahal and Dr. Roy for their continuous support, valuable advice, continuous feedback, and stimulating discussions throughout this work. In particular, working with Dr. Zdrahal on extending the Simple Time ontology was a great fun and I would especially like to thank Dr. Roy for making the visits to the CORUS Strip Products possible.

I would also like to take this opportunity to thank Prof. B. Chandrasekaran, Prof. Masahiro Hori, and Prof. Riichiro Mizoguchi for providing useful research papers on Generic Tasks, Job Assignment Library, and MULTIS and MULTIS-II system, which were crucial in formalising this research in its earlier stages.

I would also like to thank all my colleagues and friends, in particular Prof. Marc Eisenstadt at the Knowledge Media Institute for their advice, encouragement, and friendship, which were valuable in completing this work.

Finally, I would like to take this opportunity to express my forever and greatest debts to my mother and sisters for believing in me and in my dreams for all these years. Without their love, support, and encouragement this work would never have been completed. My special gratitude also goes to Siddharth and Santosh and to Anoushka and Aarya for being continuous source of happiness. Finally, I would like to express my deepest gratitude to Dr. Ashok Marathe for everything.

# List of publications

## Conference publications:

- 1) Rajpathak, D., Motta, E., and Roy, R. A Generic Task Ontology for Scheduling Applications, in the proceedings of International Conference on Artificial Intelligence'2001 (IC-AI'2001), Vol. II, (Ed.) H. R. Arabnia, CSREA press, June 25-28, 2001, Nevada, Las Vegas, USA, pp.1037-1043. ISBN: 1-892512-79-3. 2001.
- 2) Motta, E., Rajpathak, D., Zdrahal, Z., and Roy, R. The Epistemology of Scheduling Problems, in the proceedings of European Conference on Artificial Intelligence (ECAI'02), (Ed.) Frank van Harmelen, IOS press, 22nd to 26th July, 2002, Lyon, France, pp. 215-219. ISBN: 1 58603 257 7 (IOS Press), ISBN: 4 274 90525 X C3055 (Ohmsha). 2002.
- 3) Rajpathak, D., Motta, E., Zdrahal, Z., and Roy, R. A Generic Library of Problem Solving Methods for Scheduling Applications. In Proceedings of the Second International Conference on Knowledge Capture (KCAP'2003). Florida, USA, Oct. 23.25, 2003, pp. 113-120.
- 4) Rajpathak, D. and Motta, E. Poster presentation: Intelligent scheduling, at 19<sup>th</sup> workshop of the UK Planning and Scheduling special interest group, Open University, Milton Keynes, UK. 2000.

## Technical reports:

- 1) Rajpathak, D. The Task Ontology Components for Scheduling Applications. Knowledge Media Institute, The Open University, August 2001, KMI-TR-118, Technical Report. 2001.
- 2) Rajpathak, D. Intelligent Scheduling - A Literature Review. Knowledge Media Institute, The Open University, August 2001, KMI-TR-119, Technical Report. 2001.

# Contents

<b>1</b>	<b>INTRODUCTION</b>	<b>1</b>
1.1	A CASE-STUDY IN A HIGH-PRECISION MANUFACTURING SHOP	2
1.2	QUICK REVIEW OF SCHEDULING RESEARCH	5
1.2.1	Operations research and artificial intelligence	5
1.2.2	Knowledge modelling approach	5
1.2.3	Limitations of existing libraries	6
1.3	RESEARCH APPROACH	7
1.4	THESIS CONTRIBUTIONS	8
1.5	THESIS ORGANISATION	10
<b>2</b>	<b>APPROACHES TO SCHEDULING PROBLEM-SOLVING: OPERATIONS RESEARCH AND ARTIFICIAL INTELLIGENCE</b>	<b>12</b>
2.1	SCEHDULING PROBLEM TYPES	13
2.1.1	The pure scheduling problem	13
2.1.2	The resource allocation problem	15
2.2	OPERATIONS RESEARCH THREAD IN SCHEDULING	16
2.3	ARTIFICIAL INTELLIGENCE THREAD IN SCHEDULING	18
2.3.1	Basic concepts in scheduling	18
2.3.2	Constraints, preferences, and requirements	20
2.3.3	Thoery of time	22
2.4	TECHNIQUES IN ARTIFICIAL INTELLIGENCE	24
2.4.1	Constraint-based scheduling	24
2.4.2	Distributed AI: agents	25
2.4.3	Artificial neural network	26
2.4.4	Neighbourhood search methods	27
2.4.4.1	<i>Tabu search</i>	27
2.4.4.2	<i>Simulated annealing</i>	27
2.4.4.3	<i>Genetic algorithm</i>	28
2.4.4.4	<i>Fuzzy logic</i>	29
2.5	INTELLIGENT SCHEDULING SYSTEMS	30
2.5.1	ISIS	31
2.5.2	OPIS	31
2.5.3	SONIA	32
2.5.4	YAMS	33
2.5.5	FlyPast	34

2.5.6	S2	35
2.5.7	DAS	35
2.5.8	REDS	36
2.5.9	BATTLE	37
2.5.10	Summary so-far	39
2.6	DISPATCHING RULES AND HEURISTICS FOR THE JOB SELECTION	39
2.7	SCHEDULING IN A NUTSHELL	41
<b>3</b>	<b>KNOWLEDGE MODELLING APPROACHES TO SCHEDULING</b>	<b>43</b>
3.1	ONTOLOGIES AND PROBLEM-SOLVING METHODS	43
3.1.1	Ontologies	44
3.1.2	Problem-solving methods	47
3.1.3	PSMs for the scheduling task	49
3.1.3.1	<i>Propose and Revise</i>	50
3.2	EXISTING LIBRARIES FOR SCHEDULING	51
3.2.1	Hori and Yoshida's library for production scheduling	51
3.2.2	CommonKADS library	52
3.2.3	Constraint satisfaction approach for resource allocation scheduling	53
3.2.4	MULTIS-II	55
3.3	EXISTING SCHEDULING TASK ONTOLOGIES	56
3.3.1	Job assignment task ontology	56
3.3.2	OZONE	58
3.3.3	MULTIS	59
3.3.4	Summary so-far	61
3.4	LEGACY OF THE LITERATURE REVIEW: GAP ANALYSIS	62
3.4.1	Limitations in the existing scheduling libraries	62
3.4.1.1	<i>Partial coverage of knowledge-intensive methods</i>	62
3.4.1.2	<i>Domain specificity</i>	63
3.4.1.3	<i>Partial coverage to validate different areas of the scheduling task</i>	63
3.4.1.4	<i>Unsuitability for KA</i>	63
3.4.2	Limitations in the existing task ontologies of scheduling	64
3.4.2.1	<i>Insufficient degree of formalisation</i>	64
3.4.2.2	<i>Domain specificity</i>	65
3.4.2.3	<i>Commitment to specific problem-solving technique</i>	65
3.4.2.4	<i>Incomplete characterisation of the scheduling task</i>	65
3.4.2.5	<i>Incomplete validation criteria for the scheduling task</i>	66
3.5	WHAT NEEDS TO BE DONE?	66

<b>4</b>	<b>ARCHITECTURE OF THE SCHEDULING LIBRARY</b>	<b>67</b>
4.1	STATEMENT OF THE RESEARCH OBJECTIVES	67
4.2	RATIONALE FOR USING THE TMDA FRAMEWORK	68
4.3	LIBRARY ARCHITECTURE	71
4.3.1	The task component: a generic scheduling task ontology	72
4.3.2	Search as problem-solving paradigm	72
4.3.3	The method component	73
4.3.4	Development of scheduling applications from different domains	74
4.4	OCML AS A KNOWLEDGE MODELLING TOOL	75
4.5	CONCLUSION	76
<b>5</b>	<b>THE EPISTEMOLOGY OF THE SCHEDULING TASK</b>	<b>77</b>
5.1	A GENERIC SPECIFICATION OF THE SCHEDULING TASK	78
5.1.1	Validation criteria for a solution schedule	79
5.2	THE SCHEDULING TASK ONTOLOGY	79
5.2.1	Scheduling task and default schedule solution	80
5.2.2	Modelling the notion of a job	81
5.2.2.1	<i>Relations and functions required to job assignments</i>	82
5.2.2.2	<i>Relations to specify the temporal ordering among jobs</i>	83
5.2.2.3	<i>Relations to specify the job criticality</i>	84
5.2.3	Modelling the notion of a resource	85
5.2.3.1	<i>Resource-capacity axiom</i>	86
5.2.4	Modelling constraints and requirements	86
5.2.5	Representing time ranges	87
5.2.5.1	<i>Representing job and activity time ranges</i>	87
5.2.5.2	<i>Representing the schedule horizon and the resource availability</i>	87
5.2.6	Representing cost, cost function, and preference	87
5.2.7	Representing a schedule	89
5.3	COMPARISON WITH OTHER APPROACHES	89
5.3.1	Comparison with the job-assignment task ontology	90
5.3.2	Comparison with the MULTIS task ontology	90
5.3.3	Comparison with the OZONE ontology	91
5.4	CONCLUSION	92
<b>6</b>	<b>A GENERIC MODEL OF SCHEDULING PROBLEM-SOLVING</b>	<b>93</b>
6.1	SEARCH-BASED VS CONSTRAINT-BASED PROBLEM-SOLVING	93
6.2	A GENERIC SCHEDULING METHOD ONTOLOGY	94
6.2.1	Schedule space, schedule state, and schedule-state transition	94

6.2.2	Schedule operators	95
6.2.3	Job dependency network	96
6.3	A GENERIC PROBLEM-SOLVING MODEL OF SCHEDULING	97
6.3.1	The method independent control regime	97
6.3.1.1	<i>Generation and evaluation of schedule states</i>	99
6.3.1.2	<i>Schedule state selection</i>	100
6.3.2	Method specific control	101
6.3.3	Generation of a successor state: generate-new-state-successor task	103
6.3.4	Context based extension of a state: propose-schedule-from-context	104
6.3.5	Correct job selection: select-schedule-focus	105
6.3.6	Collecting and sorting the schedule operators	107
6.3.7	Resource assignment	108
6.3.8	Time-range assignment	109
6.4	COMPARISON WITH THE ALTERNATIVE APPROACHES	112
6.4.1	Comparison with the domain-specific library of production scheduling	112
6.4.2	Comparison with the constraint satisfaction approach	113
6.4.3	Comparison with CommonKADS	113
6.4.4	Comparison with MULTIS-II	114
6.5	CONCLUSION	115
<b>7</b>	<b>THE PROBLEM-SOLVING METHODS IN THE LIBRARY</b>	<b>116</b>
7.1	A GENERIC TEMPLATE TO COMPARE THE KNOWLEDGE REQUIREMENTS OF THE PSMs	116
7.2	THE SCHEDULE MODIFICATION OPERATORS	118
7.3	ENGINEERING OF THE PROBLEM-SOLVING METHODS	119
7.3.1	Hill Climbing	119
7.3.2	Propose & Backtrack	121
7.3.3	Propose & Improve	123
7.3.3.1	<i>Modelling the P&amp;I method</i>	123
7.3.3.2	<i>The control regime of P&amp;I</i>	124
7.3.3.3	<i>Foci collection and focus selection within the improve phase</i>	125
7.3.3.4	<i>Collection and selection of the improvement operators</i>	126
7.3.4	Propose & Revise	127
7.3.4.1	<i>Initial analysis of the method</i>	128
7.3.4.2	<i>Control regime of the P&amp;R method</i>	129
7.3.4.3	<i>Schedule revision</i>	130
7.3.4.4	<i>Foci collection and a focus selection</i>	131
7.3.4.5	<i>Collecting and selecting the fixes</i>	132

7.3.5	Propose & Restore-feasibility	133
7.3.5.1	<i>Modelling the P&amp;Rf method</i>	133
7.3.6	Propose & Exchange	136
7.3.6.1	<i>Initial analysis of the method</i>	136
7.3.6.2	<i>The method specific control regime of P&amp;E</i>	137
7.3.6.3	<i>Fixing the constraint violations</i>	138
7.3.6.4	<i>Foci collection and a focus selection in P&amp;E</i>	140
7.3.6.5	<i>Collection and selection of the exchange operators</i>	140
7.3.7	Propose & Genetical-Exchange (P&GE)	142
7.3.7.1	<i>Initial analysis of the method</i>	142
7.3.7.2	<i>The method specific control regime of P&amp;GE</i>	143
7.3.7.3	<i>Fixing the constraint violations in the genetical-exchange phase</i>	144
7.3.7.4	<i>Foci collection and a focus selection in P&amp;GE</i>	146
7.3.7.5	<i>The operator collection and selection in P&amp;GE</i>	146
7.4	CATEGORISATION OF THE METHODS	148
7.5	CONCLUSION	149
<b>8</b>	<b>EVALUATION STUDY OF THE LIBRARY</b>	<b>151</b>
8.1	THE SATELLITE-SCHEDULING APPLICATION	152
8.1.1	Construction of a task model	152
8.1.2	Modelling constraints and requirements	153
8.1.3	Devising a complete schedule by using Propose & Backtrack	155
8.1.3.1	<i>Construction of the operators</i>	155
8.1.3.2	<i>Focus selection knowledge and schedule construction</i>	157
8.1.4	Trying to optimise a complete schedule by Hill Climbing	157
8.1.5	Optimising a complete schedule by P&I	158
8.1.5.1	<i>Foci collection and focus selection in the improve phase</i>	158
8.2	CIPHER - A RESOURCE ALLOCATION APPLICATION	160
8.2.1	Construction of a task model	160
8.2.2	Modelling constraints and preference	163
8.2.3	Construction of a complete schedule by Propose & Backtrack	166
8.2.3.1	<i>Construction of the operators</i>	166
8.2.3.2	<i>Focus and operator selection, and schedule generation</i>	167
8.3	THE DAILY SHIP-MAINTENANCE APPLICATION	168
8.3.1	Construction of a task model	168
8.3.1.1	<i>Modelling constraints and requirements</i>	170
8.3.2	Construction of a schedule by using Generic-Schedule	172
8.3.2.1	<i>Operator construction for the daily-ship schedule</i>	172

8.3.3	Focus and operator selection	172
8.3.4	Fixing the requirement violation by using the P&Rf method	174
8.3.4.1	<i>Construction of the feasibility-restoraton operator</i>	174
8.3.4.2	<i>Construction of a feasible schedule</i>	175
8.4	THE WEEKLY SHIP-MAINTENANCE APPLICATION	175
8.4.1	Construction of a task model	175
8.4.1.1	<i>Modelling ship-maintenance jobs and resources</i>	178
8.4.1.2	<i>Modelling the constraints and requirements</i>	178
8.4.2	Applying the Propose & Backtrack method	180
8.4.2.1	<i>Construction of the operators</i>	180
8.4.3	The focus and operator selection knowledge	181
8.4.4	Modelling the propose phase	182
8.4.5	Modelling the fixes	183
8.4.5.1	<i>Fix application in the revise phase</i>	183
8.5	THE BENCHMARK APPLICATION	185
8.5.1	Construction of a task model	188
8.5.1.1	<i>Modelling the constraints</i>	189
8.5.2	Applying the Propose & Backtrack method	190
8.5.2.1	<i>Focus and operator selection</i>	191
8.5.2.2	<i>Analysis</i>	191
8.6	EVALUATING THE STATIC AND DYNAMIC PROPERTIES	193
8.7	CONCLUSION	196
<b>9</b>	<b>SUMMARY AND CONCLUDING REMARKS</b>	<b>199</b>
9.1	Summary	199
9.2	Contributions	200
9.2.1	A generic scheduling task ontology	200
9.2.2	A generic model of scheduling problem solving	200
9.2.3	A comprehensive repertoire of scheduling problem solvers	201
9.2.4	Contribution to scheduling knowledge acquisition	202
9.2.5	Contribution to scheduling epistemology	202
9.2.6	Development of job selection heuristics	202
9.3	Future research directions	203
9.3.1	Extending the current technology to develop a planning library	203
9.3.2	Interactive scheduling component	203
9.3.3	Towards nano-planning	204
	Glossary	206

REFERENCES	210
Appendix 1. A complete specification of the scheduling task ontology.	240
Appendix 2. A complete specification of the generic model of scheduling problem-solving.	252
Appendix 3. A complete specification of the simple time ontology.	274
Appendix 4. A reference guide to OCML.	283

# List of Figures

1.1	The framework of the scheduling library	8
1.2	The thesis organisation	10
2.1	Mathematical formulation of the assignment problem	17
2.2	Taxonomy of constraint, requirement, and preference	22
2.3	Architecture of OPIS (Smith, 1994)	32
2.4	Architecture of DAS	36
3.1	Taxonomy of the methods applicable to the synthesis tasks	49
4.1	Architecture of the scheduling library by instantaiting the TMDA framework	71
4.2	The search-based problem space of scheduling	73
6.1	Classification of the schedule operators	96
6.2	The complete breakdown of the method independent control regime	99
6.3	The complete breakdown of the method specific control regime	102
6.4	The complete breakdown of propose-schedule-from-context	105
6.5	The complete breakdown of Generic-Schedule	111
7.1	The evaluation-function used to calculate the job tardiness	145
7.2	Categorisation of the methods in the library	149
8.1	The order in which satellites are selected for their assignment	157
8.2	Comparison between the average CPU times required to assign the satellites	160

# List of Tables

2.1	Comparative analysis between Allen and Meng and Sullivan's time interval	23
2.2	Comparative analysis of different techniques in terms of their usability	29
2.3	Comparative analysis of intelligent scheduling systems	38
3.1	Attributes of the class job and its subclasses	57
3.2	Attributes of the class resource and its subclasses	57
3.3	Main components in the OZONE ontology	58
3.4	Comparison between different task ontologies	62
5.1	The job-specific relations and functions	82
6.1	Different methods to select a schedule state	100
7.1	The knowledge requirements of Generic-Schedule	117
7.2	The knowledge requirements of the hill climbing method	120
7.3	The knowledge requirements of the P&B method	122
7.4	The knowledge requirements of the P&I method	126
7.5	The knowledge requirements of the P&R method	132
7.6	The knowledge requirements of the P&Rf method	135
7.7	The knowledge requirements of the P&E method	141
7.8	The knowledge requirements of the P&GE method	147
8.1	Properties of the scheduling applications	151
8.2	The capacity distribution of all the project-staffs	161
8.3	The resource requirement of each work-package	161
8.4	Data used to formalise the ship-maintenance jobs	168
8.5	Maximum capacity of the ship-maintenance resources	169
8.6	Data used to formalise the ship-maintenance jobs	176
8.7	The precedence relation among work-steps	186
8.8	The capacity of the zone and labour resources	187
8.9	The duration and resource requirement of all the work-steps	188
8.10	Comparison between the performance of scheduling applications	192
8.11	Summary of the evaluation of the static and dynamic properties	194

# Chapter 1

## INTRODUCTION

In this thesis we propose a library of reusable components for building problem-solvers for scheduling.

Scheduling is the central theme of our thesis. As a first ‘high-level’ approximation, we can say that the scheduling task deals with *the assignment of jobs and activities to resources within a specific time range in accordance with relevant constraints and requirements*. Scheduling is a *decision making* process in today’s industry. Typical domains include: manufacturing scheduling, project scheduling, resource allocation scheduling, transportation scheduling, mass transit scheduling, scheduling nurse shifts in hospital, air gate assignment scheduling, hydropower scheduling, and so forth. This list is by no means an exhaustive one, but gives an idea of the ubiquity of the scheduling task. Each scheduling domain imposes its unique constraints and requirements, which must be obeyed by a scheduler while devising a schedule, because they determine the space of a valid solution. A process of constructing a schedule becomes even more challenging due to the *uncertain, dynamic, and unpredictable* circumstances that occur in an environment where the scheduling task has to be carried out (Fox and Kempf, 1985). For instance, in a manufacturing scheduling environment new orders come continuously, which take priority over the existing ones, and therefore, the existing schedule may need to be revised. To come up with a good quality schedule in an uncertain environment is a highly creative activity. A scheduler needs to acquire systematic knowledge about the various events that might take place in a scheduling environment.

The term ‘knowledge’ that is used in the preceding paragraph and which will be understood in this thesis can be conceived as having the following three implications (Nickols, 2000): a) it represents the state of an agent (either human or artificial) which is aware of the facts, methods, rules, axioms, and techniques of an environment within which it operates; b) it indicates a competence like notion, the ability of an agent which is capable of executing rational actions to reach a solution; and c) it can be captured and acquired from experts and codified in a computational system. Nonaka and Takeuchi (1995) emphasises that the collective intellectual knowledge of a firm can be considered as a ‘strategic resource’ of the firm.

Having briefly introduced the two main components of this thesis, in the following section we describe a case study of a high precision machine shop, which aims at

highlighting a need for considering the scheduling task as a serious activity in a complex environment of a modern industry to smoothen its operation. Moreover, it also emphasises the fact that wherever possible the system components must be made as reusable as possible to avoid their brittle nature. In section 1.2 we will provide a brief overview of existing research in the scheduling area and highlight the limitations of the existing scheduling libraries. In section 1.3, we will outline our approach to library construction showing how we approach the limitations of the existing scheduling libraries. In section 1.4, we will then briefly describe the main contributions of our research. Finally, in section 1.5, we conclude the chapter by outlining the organisation of our thesis.

## **1.1 A case-study in a high-precision manufacturing shop**

Here, we present a small case-study (McKay, Safayani, and Buzacott, 1988) that will explicate the importance of the scheduling task.

The environment we describe here is a large machine shop that manufactures high-precision components for the aerospace industry. Each high-precision component has approximately 80 operations, which need to be performed over 300 different work-stations, with an average of 5,000 open work orders. An initial complexity in this environment is imposed by the size of the application alone, and thus the need for computational support is significant. However, this is not the end of the story; there are various other factors that add complexity to this environment.

The set-up and processing times for manufacturing each precision component vary in time. That is to say, the time required to produce a component on the same machine with the same resource can vary from 3 to 6 days. Therefore, the due date of all other components that depend on the current component may need to be updated according to the changes in the current operations. Moreover, the processing times of the components within each batch change almost every time primarily due to unpredictable organisational factors like unavailability of the resources or machine failure. Such unforeseen events make the prediction and forecasting difficult.

The manufacturing of the components can be pre-empted at any time and can start somewhere else on the time line. Naturally, the temporal order among components needs to be amended in compliance with the changes introduced by a pre-empted component. At times the importance of the components changes dynamically according to changes that may occur in the manufacturing conditions in the aerospace industry. Consequently, components with an increased importance need to be pushed forward to fulfil their new due dates. Moreover, the top-level management may favour rush orders which must be

accommodated in the existing batch for their accomplishment. These last minute orders can make the existing components become late. The raw material may fail to arrive in time because of the wrong forecasting performed by the inventory-management department. Even when the resources arrive on time, the existing resources may become unavailable during the critical stages of manufacturing. In some cases, the components have to be re-directed to alternative resources or in the worst-case scenario they cannot be executed and therefore miss their dispatching dates. The final complexity is introduced by the shop's atmospheric conditions. Because these are high-precision components, the machines on which the manufacturing of these components take place are high-precision machines, which are very sensitive to the atmospheric conditions. Even a small change of a few degrees in the shop's temperature may be enough to throw the components out of precision.

This particular shop works seven days a week in three shifts to meet the demand levels placed by customers. Moreover, almost each member of personnel in each shift works overtime. Nevertheless the shop often fails to meet the required throughput. In a nutshell, this shop exhibits different types of complexities that can be observed in the real world. It can be envisaged that a stable scheduling system that can take into consideration all the intricacies in the manufacturing environment is necessary for the smooth operation of this shop.

Analogous to the manufacturing shop discussed in this section, other domains of scheduling also have their own complexities that differ in nature, and a sound scheduling system is important for their smooth performance. A quote from the working research papers of NASA exemplifies the importance of the scheduling task in space operations:

*"Operations on most U.S. manned space missions, including Space Shuttle/Spacelab flights, are scheduled in great detail long before launch"* (Maxwell and Howell, 1995).

Obviously, to construct a scheduling system that deals with different issues involved in a scheduling domain is a challenging task, which requires a substantial amount of investment. Usually, a scheduler (here we refer to a scheduler as a human agent) acquires his/her vast amount of expertise through years of experience and practice, and such a repertoire of expertise forms his/her 'knowledge-base'. This allows a scheduler to devise a good quality schedule by tackling all the complexities that are involved in a scheduling environment. Obviously, a natural goal here would be to determine the extent to which such a high level dependency on human expertise can be reduced by means of a computational system.

Having formalised a scheduler's knowledge in a computational system in order to reduce human dependency, another important issue needs to be tackled, which deals with determining the level of reusability system components must attain. It is a serious question because failing to tackle this question would result in developing a system that subscribes to a specific scheduling domain that becomes obsolete quickly, mainly because of its incapability of handling specifications from a different scheduling domain. Various techniques can be used to elicit and acquire knowledge from human experts which in turn help in constructing the reusable components to tackle the scheduling task across different domains. However, this very process of acquiring and representing expert knowledge has traditionally been considered as a bottleneck activity (Gaines and Shaw, 1993). A computational system which by making use of its knowledge-base reaches a solution to a problem can then be referred to as a *knowledge-based system* (KBS).

Here, we need to make a decision about how to represent such acquired knowledge into a computer system. The main reason why such a decision needs to be taken is because, as has been pointed out by Steels (1990), there is an observable gap between the knowledge and problem-solving expertise observed in the human experts and the implementation level. Newell (1982) in his cornerstone article 'The Knowledge Level' has already proposed an answer to this question by formulating '*the knowledge-level hypothesis*':

*"Knowledge is to be characterised entirely functionally, in terms of what it does and not structurally, in terms of physical objects with particular properties and relations".*

In the same article, Newell proposed the '*principle of rationality*', which postulates the rational problem-solving behaviour of an agent:

*"If the agent has the knowledge about choosing particular action among the several available actions, which can lead towards the solution or goal state then the agent will choose that particular action".*

Consistently with Newell's proposal, and in line with the work by Motta (1999), Steels (1990), and Breuker and Wielinga (1985), etc. in this thesis we will follow '*the knowledge modelling approach*' to KBS construction. Thus knowledge will be systematically represented at the knowledge level independently of its physical realisation in a computational system. Another aspect of the knowledge modelling paradigm is that the knowledge acquisition (KA) process is driven by pre-existing knowledge models, often represented as *ontologies* (Gruber, 1995). More importantly, as pointed out by Motta (2001), the KA approach to the system construction has following advantages: the discrete pieces of knowledge can be elicited from a domain expert and encoded in a computational

system and therefore a virtual domain expert can be constructed that can replicate the problem-solving expertise of a human domain expert. Finally, the cognitive perspective of the KA theory as suggested by Newell and Simon (1976) and Newell (1982), proposed a production system to describe a general intelligent behaviour in a problem space.

In the following section, we provide a brief background of research in the field of scheduling. More detailed literature review will be presented in Chapters 2 and 3.

## 1.2 Quick overview of scheduling research

### 1.2.1 Operations research and artificial intelligence

Scheduling is a meticulously researched area in Management Science and Operations Research (OR) (Conway *et al.*, 1967; Baker, 1974; French, 1982). Although, the classical techniques of OR have proposed sophisticated mathematical models and algorithms, these efforts have shown limited applicability when implemented in real-life applications, as they cannot handle heterogeneous resources and their rigid and static formulation fails to provide enough leverage to handle the dynamicity present in the real-world.

Scheduling has also garnered serious attention from AI researchers. Fox and his group in the 1980s started developing the first intelligent scheduling system called ISIS and in later years several intelligent scheduling systems have emerged (Prosser and Buchanan, 1994) to tackle scheduling problems in different domains. Although these systems have exploited various techniques in AI successfully their major drawback was domain specificity, which restricted the reusability of these systems within a single application domain. Consequently, a new system had to be built from scratch for each domain.

### 1.2.2 Knowledge modelling approach

Reusability is the main concern of research in knowledge modelling. Here, the construction of a KBS can be realised by applying libraries of problem-solving methods (PSMs) (Motta, 1999; Breuker and van de Velde, 1994). An Ontology (Gruber, 1995) and a PSM (Gomez-Perez and Benjamins, 1999) are the two most central components in the construction of a library. These two components are instrumental particularly because of their ability to enhance the *sharing* and *reusability* of system components over wider domains. A PSM can either be task specific or task independent. Task specific PSMs are developed to tackle specific types of Generic Tasks (Chandrasekaran, 1986), such as planning, parametric design, diagnosis, assignment, and so on. Task independent PSMs do not subscribe to any particular task, but rather provide reasoning steps in terms of a generic paradigm, such as search (Newell and Simon, 1976).

As discussed in Wielinga and Schreiber (1997) various knowledge-intensive PSMs<sup>1</sup> have been developed to tackle classes of synthesis tasks, such as design, planning, assignment, scheduling. Some influential examples include Propose & Backtrack (Runkel *et al.*, 1996), Propose & Improve (Motta, 1999), Propose & Revise (Marcus and McDermott, 1989), Propose & Exchange (Poeck and Puppe, 1992), and so on.

A knowledge modelling framework provides a methodology to organise the different building-blocks of a library. Moreover, it also specifies how the different components of a library are related to each other. Some of the influential knowledge modelling frameworks developed in the field are Generic Tasks Structures (Bylander and Chandrasekaran, 1988; Chandrasekaran *et al.*, 1992), Role-Limiting Methods (Marcus, 1988), Protégé-II (Musen *et al.*, 1993), CommonKADS (Schreiber *et al.*, 1994), MIKE (Angele *et al.*, 1998), Components of Expertise (Steels, 1990), EXPECT (Swartout and Gil, 1995), GDM (Terpstra *et al.*, 1993), and Task-Method-Domain-Application (TMDA) (Motta, 1999).

Based on the knowledge modelling frameworks enumerated above various task-specific libraries have been constructed. Some examples of task-specific libraries include diagnosis (Benjamins, 1995), parametric design (Motta and Zdrahal, 1996), planning (Valente *et al.*, 1998); assessment (Valente and Löckenhoff, 1993), etc. *The research conducted in this thesis subscribes to this stream where our aim is to construct a generic library of scheduling PSMs.*

### 1.2.3 Limitations of existing libraries

In the field of scheduling, various attempts have been made in the past at constructing libraries (Hori and Yoshida, 1998; Sundin, 1994; Tijerino and Mizoguchi, 1993; Le Pape, 1994). However, these earlier attempts have failed to provide comprehensive results mainly because of the following reasons:

- **Partial coverage of knowledge-intensive methods:** Existing libraries for scheduling provide either very little or no coverage at all for the knowledge-intensive PSMs to tackle the scheduling task. For instance, the CommonKADS library for assignment and scheduling tasks (Sundin, 1994) only comprises the Propose & Revise method.

---

<sup>1</sup> By “knowledge-intensive” we mean that these PSMs make heavy use of the application domain knowledge in order to improve their reasoning efficiency. For instance, the Propose & Revise method (Marcus and McDermott, 1989) relies on the application domain knowledge to determine how the constraints that are violated while constructing a schedule can be fixed by proposing a new set of assignments for the jobs involved in conflict.

- **Domain specificity:** Existing libraries for scheduling subscribe to specific scheduling domains, and therefore, the reusability of these libraries is limited. For instance, Hori and Yoshida's library (1998) subscribes to the domain of production scheduling. Therefore, all the problem-solvers from their library are developed in such a way that they can only be used to tackle production scheduling.
- **Partial coverage to validate different areas of the scheduling task:** As described in the first bullet point, because the existing scheduling libraries fail to provide a comprehensive coverage of the different knowledge-intensive PSMs, the problem-solvers from these libraries also fail to provide a comprehensive coverage of the different types of scheduling tasks. Generally speaking, these libraries validate the scheduling task only against completion and constraint violation, but they fail to cover requirement violation and optimisation issues.
- **Unsuitability for knowledge acquisition:** Some of the existing libraries subscribe to a specific problem-solving technique. For instance, ILOG SCHEDULER of Le Pape (1994) subscribes to constraint-satisfaction as its problem-solving technique. Given this uniform approach to modelling as constraint-satisfaction, it does not provide a good enough 'epistemological' framework to analyse the different knowledge-intensive tasks and methods that take place while constructing a schedule.

In the following section, we outline our research approach to construct a library of scheduling PSMs.

### 1.3 Research approach

In order to bridge the aforementioned problems with existing libraries of scheduling, in this thesis we aim to develop a task-specific, but domain independent library of scheduling PSMs. In our approach, we subscribe to the TMDA knowledge modelling framework (Motta, 1999), which provides a methodology to organise our library. A more detailed discussion for subscribing to the TMDA knowledge modelling can be found in Chapter 4 (cf. Section 4.2). The entire library will be formalised by using the Operational Conceptual Modelling Language (OCML) (Motta, 1999). Figure 1.1 depicts a simplified version of our library architecture, the more detailed framework can be found in Chapter 4.

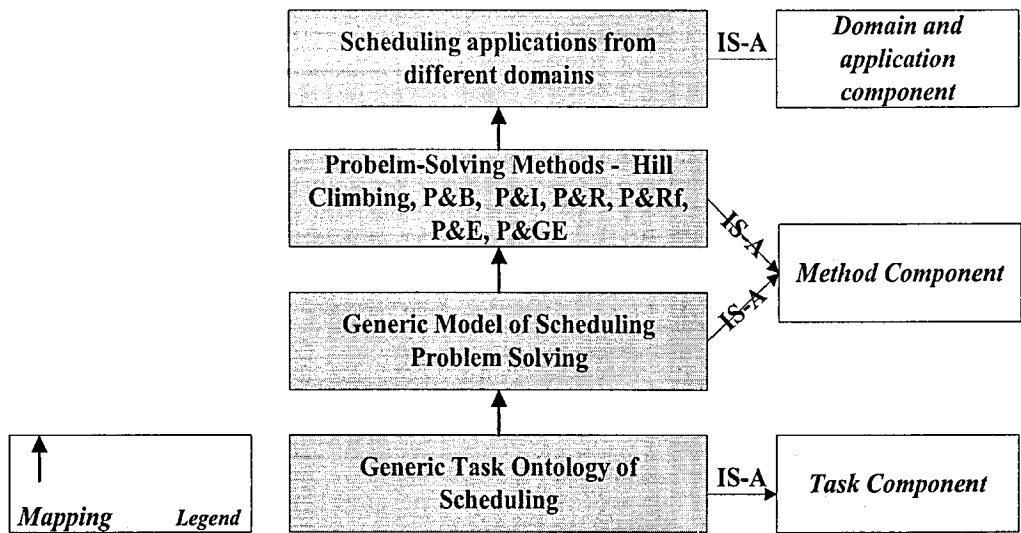


Figure 1.1. The framework of the scheduling library.

The different building-blocks of our library are described in the following bullet points:

- **Task component:** It is the first building-block of our library where we formalise the nature of the scheduling problem in terms of a generic task ontology. The task ontology is generic because it does not subscribe to any particular application domain or reasoning method.
- **Method component:** It is the second building-block of the library, which provides the reasoning service of the library. As it can be observed from Figure 1.1, the method component of our library is divided into two sub-components: a generic model of scheduling problem-solving and the different PSMs. The former is a constructive component of the library that takes as an input the scheduling task ontology and then subscribes to the search problem-solving mechanism. It then provides a detailed breakdown of the main subtasks and methods (PSMs) for building complete problem-solvers for scheduling. At the next level we develop more specialised knowledge-intensive PSMs by reusing and specialising the high-level tasks included in the generic model of scheduling problem-solving. The PSMs in our library are constructed in such a way that they cover and reason about all the validation areas of scheduling.
- **Domain and application components:** These are the last two components of our library. At this stage, we validate the generic nature of our library by constructing scheduling applications from different domains.

## 1.4 Thesis contributions

The following main contributions can be drawn from our thesis.

- *A generic task ontology for scheduling:* Existing scheduling task ontologies (Hama *et al.*, 1992a, b; Mizoguchi *et al.*, 1995; Smith and Becker, 1997) provide limited results

because in some cases they subscribe to a specific application domain or in some other cases they subscribe to specific ‘problem-solving shells’. More importantly, crucial ontological distinctions are typically missing from their underlying frameworks. Our task ontology overcomes these shortcomings by providing a user with computational model of scheduling that can be reused to acquire scheduling knowledge from a variety of domains;

- ***A generic model of scheduling problem-solving:*** Existing libraries of scheduling fail to provide a clear separation between the reusable high-level components and the non-reusable components. Consequently, it becomes difficult to realise how the reusability of library components can be exploited to construct a new PSM. Our generic model of scheduling problem-solving component overcomes this problem by providing a high-level repertoire of reusable tasks and methods. As a result, it allows us to construct a new PSM simply by its reuse and specialisation<sup>2</sup>;
- ***Comprehensive repertoire of scheduling problem-solving methods:*** Ours is the first library in the field that provides a comprehensive coverage of PSMs that can be used to tackle the different types of scheduling task;
- ***Contribution to KA in scheduling domain:*** Throughout this thesis we will construct various templates either in the form of different ontologies, such as task ontology and method ontology, or as generic templates that can be used to compare and contrast the knowledge requirements of different PSMs. These generic templates can be used to acquire the relevant scheduling knowledge through their instantiation. Here, the term ‘knowledge acquisition’ is used to represent a theoretical knowledge engineering activity necessary to acquire the problem-solving knowledge needed to execute the reasoning process;
- ***Contribution to the epistemology of the scheduling task:*** Our scheduling task ontology is based on a clear theoretical model of the scheduling task. This theoretical model distinguishes between components, such as constraints, requirements, and

---

<sup>2</sup> For instance, as it will be shown in Chapter 7, a new PSM can be constructed quickly by subscribing to the method specific control regime called `expand-incomplete-state` (cf. Chapter 6, section 6.3.2) of `Generic-Schedule` and only those tasks will be newly defined that tackle the constraint and requirement violations and optimisation issues. E.g., in *Propose & Revise* (Marcus and McDermott, 1989) a new task called `revise-schedule` (cf. Section 7.3.4.2) is defined newly in `expand-incomplete-state` control regime in order to tackle the constraint violations, or the foci (i.e., constraint violations) in the revise phase are collected by defining a new method called `collect-all-constraint-violations` (cf. Section 7.3.4.4), which achieves the task `collect-state-foci` from `Generic-Schedule`.

preferences, which play a crucial role in validating a solution schedule. They are rather sloppily distinguished in the existing proposals (Hama *et al.*, 1992a, b; Mizoguchi *et al.*, 1995; Smith and Becker, 1997), if distinguished at all. Moreover, at the method level the generic model of scheduling problem-solving and the different PSMs provides a useful insight into the various tasks and methods that are crucial for constructing a schedule;

- **Development of new job-selection heuristics:** While developing a generic model of scheduling problem-solving, we also developed three new job-selection heuristics that improve the selection of the correct candidate job and as a result improve the efficiency of a schedule construction. These heuristics were derived from the real-life scheduling scenario.

1.5 Thesis organisation

In this section, we describe how all the chapters in our thesis are organised. Figure 1.2 depicts the flow of the chapters in our thesis.

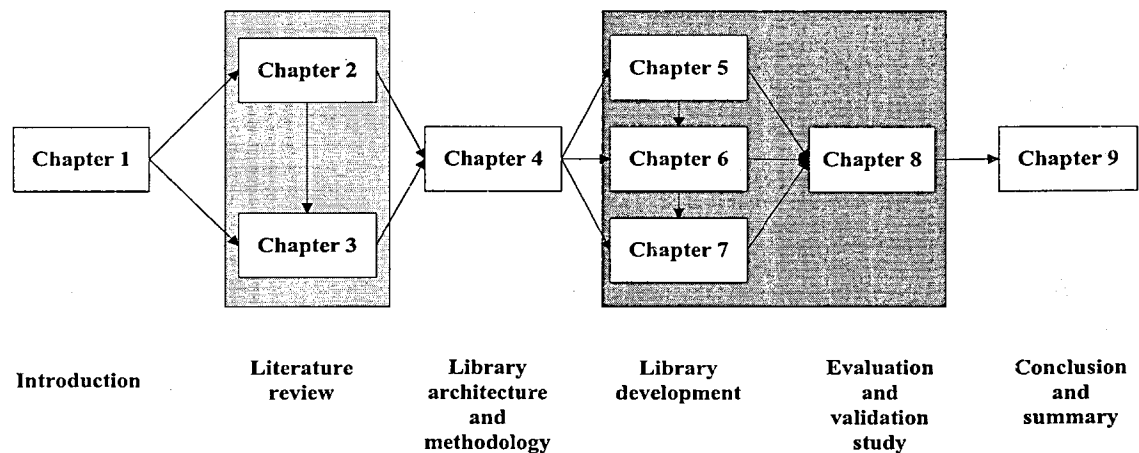


Figure 1.2. The thesis organisation.

Chapter 2 provides an overview of the different streams of research, such as OR and AI involved in the scheduling area. While talking about the AI approaches to scheduling we first discuss the different techniques that can be used to tackle the scheduling task and then provide a review of the different intelligent scheduling systems that were developed in the 1980s and 1990s. This chapter is particularly important in understanding the theoretical foundation that underlies the scheduling task. In Chapter 3, we describe the knowledge modelling approaches to library construction. This literature review is directly relevant to our thesis, whereby first we acquaint ourselves with the two most central concepts for constructing a library of knowledge-level components, i.e. ontology and PSM. Having done this, we review the status of the existing scheduling libraries and the scheduling task ontologies. Based on this part of the literature review, we highlight the major shortcomings

in the existing approaches. These shortcomings allow us to formulate the specific objectives of our research. Finally, we conclude this chapter by providing an insight into how we will approach to overcome the shortcomings observed in existing scheduling libraries.

In Chapter 4, we describe the architecture and organisation of our library. Our library organisation can be understood by the TMDA knowledge modelling framework. Then we describe how different components in our library are interrelated with each other. Finally, we introduce the OCML knowledge modelling language, which will be used to implement our library.

In Chapter 5, we describe the first building-block of our library, which can be realised by the generic scheduling task ontology. First, we describe a generic theoretical framework to frame the scheduling task, and then we describe all the important modelling decisions taken while developing the task ontology.

In Chapter 6, we describe the second building-block of our library, which is the generic model of scheduling problem-solving. Here, we first describe a generic method ontology necessary to characterise search based problem-solving behaviour of the scheduling task. Then we describe all the tasks and methods developed to construct the generic model of scheduling problem-solving. In Chapter 7, we describe the second part of the method component by discussing how all the PSMs in our library have been engineered by reusing high-level tasks and methods developed in a generic model of scheduling problem-solving.

In Chapter 8, we describe the evaluation study of our library conducted on five real-life and benchmark scheduling applications in order to confirm its generic nature.

In Chapter 9, we conclude our thesis by first summarising the research conducted in our thesis and then by discussing in further detail the main contributions that can be drawn from our research. Finally, we conclude our thesis by providing an insight into our future research directions.

# Chapter 2

## APPROACHES TO SCHEDULING PROBLEM-SOLVING: OPERATIONS RESEARCH AND ARTIFICIAL INTELLIGENCE<sup>1</sup>

The literature review presented in this chapter highlights the research approaches that have been evolved over the years in scheduling and related fields. In this chapter we will do more than simply summarising these past efforts and we will provide a roadmap for i) the different models that have been developed to characterise the scheduling task, ii) the techniques that can be used to solve scheduling problems, and iii) different intelligent scheduling systems that have been developed over the period of last two decades. Then we also provide an overview of the various heuristics, which have been devised to select a correct job to improve the efficiency of schedule construction.

To tackle the scheduling task various streams of research have emerged both in Operations Research (OR) and Artificial Intelligence (AI), which will be reviewed in this chapter. The content of the chapter is organised as follows. In the following section we will review different problem types that have been developed to characterise the scheduling task. Then in section 2.2 and 2.3, we will provide an overview of OR-based and AI-based approaches to scheduling. In section 2.3.1, we will discuss commonly found concepts in scheduling and then in section 2.3.2 we will highlight some key notions, such as constraints, requirements, and preferences and their role in validating a solution schedule. Then in section 2.3.3, we will analyse different formalisms that have been put forward to conceptualise the time element in scheduling. Having reviewed the components of scheduling, in section 2.4, we will review various techniques developed over the years to tackle the scheduling task, and then in section 2.5 we will review various intelligent scheduling systems which have been constructed by using the techniques from AI. In section 2.6, we will review different dispatching rules and heuristics developed in OR and AI for efficient job selection. Finally, in section 2.7 we conclude our chapter by summarising main results from the review conducted here.

---

<sup>1</sup> Here we use the term Artificial Intelligence to refer to those approaches, which use the traditional AI techniques, but do not subscribe to the *knowledge modelling* approach. Given that knowledge modelling research can be seen as a part of AI, it is important to realise that this distinction is purely pragmatic and does not carry any deep epistemological meaning.

## 2.1 Scheduling problem types

At an abstract level the scheduling problem can be classified into the following three problem types: *the pure scheduling problem*, *the resource allocation problem*, and *the joint scheduling problem*. Analysis of these problem types is important for us as it enables us to identify the unique features associated with these models, which we want to subsume in a generic scheduling task ontology. Here, we will concentrate our discussion on the first two problem types, mainly because these categories have become standard examples in the scheduling community much like the blocks world has become a standard example in the planning community. The third problem type can be easily constructed as the combination of the pure scheduling problem and the resource allocation problem.

### 2.1.1 The pure scheduling problem

The pure scheduling problem characterises scheduling from the viewpoint of a manufacturing scheduling environment. In this environment, the pure scheduling problem is classified into three sub-groups:  $\alpha | \beta | \chi$  (Lawler, 1983), where  $\alpha$  indicates the number of resources and also specifies a similarity feature among them (e.g., homogeneous machines),  $\beta$  indicates the job characteristics (period, deadline, precedence constraints), and  $\chi$  indicates the notion of optimality. The pure scheduling problem can further be classified into the *deterministic job-shop scheduling (JSS) model*. JSS is the most classical model of scheduling and the other models of scheduling, such as open-shop, flow-shop, and mixed-shop can be derived from it.

Jackson (1956) generalised Johnson's flow-shop algorithm (1954), which consists of  $n$  jobs and  $m$  machines along with the temporal precedence relations among jobs. Over the years it became a kind of standard format to represent the JSS model. JSS is one of the widely studied areas in scheduling. Below we characterise the nature of JSS.

A JSS model can be represented by a set of jobs,  $J = \{j_1, \dots, j_n\}$  and a given set of physical resources  $R = \{r_1, \dots, r_n\}$ . Each job  $j_1$  consisted of a set of operations (also referred to as activities) that can be indexed as  $O_j = \{o_{j1}, \dots, o_{jn}\}$ . For instance, in the manufacturing environment, a drilling job could have operations such as: drilling-machine set-up, loading of a drilling job on a drilling-machine, actual drilling operation, and unloading of a drilling job from a drilling-machine. The jobs and the operations can be assigned over resources in accordance with a process routing, which specifies a partial ordering among them. For instance, the temporal precedence relations among any two jobs can be of the form, job<sub>1</sub> *BEFORE* job<sub>2</sub>, job<sub>1</sub> *AFTER* job<sub>2</sub>, etc. A job, say  $j_1$  has a specific release and due date associated with it, which can be represented by  $rd_{j_1}$ , and  $dd_{j_1}$

respectively. A job  $j_1$  must complete its execution between these two dates. Each job also has a fixed duration,  $d_{j_1}$ , and a variable state time,  $st_{j_1}$ , associated with it. The domain of all such possible start times can be constrained by specifying the release date of a job, which can be represented as:  $(1 \leq st_{j_1} \leq rd_{j_1})$ . For the successful completion of a job and its operations, jobs and operations can be assigned over  $r_i$  different resources (e.g., machines, personnel, etc.), and the domain of all such resources can be represented by,  $R_{ij}$  ( $1 \leq j_1 \leq r_i$ ). Also, each job can have a pool of resources and a resource that needs to be assigned to a job can be chosen from this pool. Such a resource pool can be represented as,  $\Omega_{ij} = \{r_{ij1}, \dots, r_{ijn}\}$ , where  $r_{ijn} \in R_{ij}$ . The following types of constraints can usually be found in JSS:

- **Functional constraints:** Limit the types of jobs and operations each resource can process at any given time in a schedule depending upon the functionality of a resource. For instance, the milling machines can perform only milling type of jobs;
- **Capacity constraints:** Restrict the number of jobs each unit capacity resource can handle at any given time in a schedule. A capacity constraint of a unit capacity resource can be translated into the following disjunctive constraint:  $(\forall j_1 \forall j_2 (R_{j_1} \neq R_{j_2}) \vee et_{j_1} \leq st_{j_2} \vee et_{j_2} \leq st_{j_1})$ , where  $et_{j_1}$  and  $et_{j_2}$  represents the end time of the jobs  $j_1$  and  $j_2$  respectively. This states that two jobs, say  $j_1$  and  $j_2$ , cannot share the same resource for their execution; otherwise, their time ranges cannot overlap with each other. A resource capacity conflict among any two jobs can be avoided by imposing a precedence relation among start and end times of conflicting jobs;
- **Availability constraints:** Specify when a particular resource is available for accomplishing the assigned jobs. All the jobs must obey the availability period of a resource on which they are assigned;
- **Precedence constraints:** Specify a job processing routing among any two jobs. They can be translated into linear inequality of the form:  $et_{j_1} \leq st_{j_2}$ , then  $j_1$  *BEFORE*  $j_2$ , where  $et_{j_1} = st_{j_1} + d_{j_1}$ .

A more detailed review of JSS can be found in Jain and Meeran (1998) and Jones and Rabelo (1998). The open-shop scheduling (OSS) model (Dorndorf *et al.*, 2000) can be derived from JSS. The main difference between JSS and OSS is that the latter imposes no specific ordering constraint over the execution of jobs and operations. On the other hand, in the flow-shop scheduling model each job has exactly one operation associated with it

that needs to be assigned to the resource. All the jobs and operations must go through all the machines in the same order.

### 2.1.2 The resource allocation problem

In comparison with JSS, the resource allocation problem is more deterministic in nature because a scheduler has prior knowledge about the demand for resources in order to process jobs and operations. The problem subscribes to a resource-based scheduling perspective (Brusoni *et al.*, 1996) by assigning resources to jobs in time. Usually, the resource allocation problem can be formalised based on the factory scheduling perspective (Fox and Sadeh, 1990). Talbot (1982) presents a general class of the non pre-emptive resource-constrained scheduling problem, in which the quality of a schedule is measured based on an evaluation function. Typical examples of an evaluation function include maximisation of resource utilisation, minimisation of the consumption of critical resources, minimisation of operational cost, and penalties, etc. Gudes *et al.* (1990) presents a general paradigm for solving the family of resource allocation problems. Typical applications of the resource allocation scheduling include air-gate assignment and room allocation (Smith *et al.*, 2000). The resource allocation problem can further be classified into *a single resource scheduling* and *a multiple alternative resource scheduling*.

Single resource scheduling involves a single indivisible resource that needs to be assigned over time to '*n*' jobs and '*o*' operations. The *jobs* and *operations* have equal duration and they are unrelated to each other, i.e., no precedence relation exists among them. The problem can be represented as follows: there exist *n* jobs  $\{N_1, \dots, N_n\}$  and each job can have  $o_n$  operations associated with it and they are represented as,  $O_{Ni} \in \{1, \dots, n\}$ . The main aim of the problem is to assign a resource to each job and operation in compliance with the following two constraints:  $\forall ij [(i \neq j) \supset (N_i \neq N_j)]$  and over operations  $\forall ij [(i \neq j) \supset (O_{Ni} \neq O_{Nj})]$ . The former constraint states that no two distinct jobs can occupy the same type of resource at a same time, whereas the latter constraint imposes the same condition on the assignment of operations. These two constraints are similar in spirit to that of the *capacity constraint* from JSS.

The multiple alternative resource scheduling is more realistic in nature, compared to the single resource problem. In this problem one or more resources can be assigned to jobs and operations, and therefore, this problem augments the level of complexity of the assignment. The problem can be described as follows: for *n* jobs and  $o_n$  operations associated with *n* jobs a schedule must choose one of the *m* resources that can be assigned to accomplish the execution of jobs and operations. A formal representation of the

assignment can be given as follows:  $N_i \in \{ \langle T_i, R_i \rangle \mid 1 \leq T_i \leq n, 1 \leq R_i \leq m \}$ , where  $T_i$  and  $R_i$  represent a time range and a resource assigned to a job  $n$ . The similar representation can be used to indicate an assignment of operations within each job. Each job has an associated due date, say  $d_n$ , which can be indexed as  $d_{ni} \in \{1, \dots, n\}$ . Finally, all the jobs must maintain a precedence relation say  $Pr_{ij}$  among them, where  $Pr_{ij} = 1$  if  $j_i$  precedes  $j_j$ .

The multiple alternative resource scheduling imposes the following types of constraints (Fox and Sadeh, 1990):

- A job  $j_i$  must end on or before its due date represented by,  $\forall j_i [T_i \leq d_{ji}]$ , where  $T_i$  represents the time interval of  $j_i$  and  $d_{ji}$  represents the due date of  $j_i$ . In other words every time point, say  $T_{pi}$  in  $T_i$  is less than or equal to  $d_{ji}$ ;
- Any two jobs  $j_i$  and  $j_k$  must maintain the precedence relation among them if imposed, such that  $j_i$  must be assigned before  $j_k$  represented by,  $\forall j_i j_k [(P_{ik} = 1) \supset (T_i < T_k)]$ ;
- No two jobs using the same resource may occupy the same time slot in a schedule, which is represented as,  $\forall j_i j_j [((i \neq j) \wedge (R_i = R_j)) \supset (T_i \neq T_j)]$ .

Here, we conclude our discussion about the problem types for framing the scheduling task. These problem types highlight the different features that need to be taken into account while characterising the scheduling task. In the following section we discuss the research involved in the OR domain.

## 2.2 Operations research thread in scheduling

In OR, the classical approaches to scheduling are characterised by a reduction of the scheduling problem into the formulation of assignment and sequencing problem. However, a few critical differences exist between these problem types as discussed below.

The fundamental difference between the assignment and the scheduling problem concerns the allocation of resources to parameters. The assignment problem can be characterised by two sets of objects: *demand* and *supply*, where each element of the former set must be assigned over the latter set (Hillier and Libermann, 1974). In contrast with the assignment problem, the scheduling problem not only assigns jobs to resources, but also fixes a time range for its accomplishment. These two problem types can also be distinguished based on the way they characterise the time line. While scheduling takes place over a discrete time line (Bartak, 1999), the nature of a time line is usually considered to be a continuous one in assignment problems (Hillier and Libermann, 1974). In this sense, assignment problems are a particular case of scheduling problems, because in

scheduling one can jump from one time point to another and this jumping is not allowed in assignment. However, as argued by Liu (1988) and Lloyd (1982), in real-life domains it is hardly possible to assure the continuous nature of a time line, because jobs often do get perturbed during their execution and they start at some other point in time. Finally, an additional problem with reducing scheduling problems to assignment ones is that the techniques that have been developed to deal with the assignment problem fail to handle the heterogeneity of the resources.

In comparison with the assignment problem, the sequencing problem simply determines the order in which jobs need to be processed on a particular resource (French, 1982). De Werra (1985) has proposed a new technique to derive a schedule based on the formulation of the sequencing problem, which is referred to as the time-tabling problem. The following box represents a mathematical formulation of the assignment problem in OR (Sharma, 1998).

$$\begin{aligned} \text{Minimise Total Cost: } Z &= \sum_{i=1}^n \sum_{j=1}^n C_{ij} \cdot X_{ij}, i = 1, 2, \dots, n; j = 1, 2, \dots, n \\ X_{ij} &= 1 \text{ if } i^{\text{th}} \text{ job is assigned to } j^{\text{th}} \text{ resource, } 0 \text{ otherwise;} \\ \sum_{i=1}^n X_{ij} &= 1 \text{ (one job is done by the } i^{\text{th}} \text{ resource, } i = 1, 2, \dots, n) \\ \sum_{j=1}^n X_{ij} &= 1 \text{ (only one resource should be assigned the } j^{\text{th}} \text{ job, } j = 1, 2, \dots, n). \end{aligned}$$

Figure 2.1. Mathematical formulation of the assignment problem.

As it can be observed in the above definition, optimisation is the central theme in the formulation of the problem in OR; however optimisation normally suffers from the combinatorial complexity that can be proved NP-hard (Garey and Johnson, 1979). Not surprising, Prosser (1989) argued that the optimisation aspect of assignment is difficult if not impossible to achieve in real-life. Nevertheless, George Dantzing developed two techniques: linear programming (LP) and the simplex method, which effectively tackled optimisation. Other formulations, such as the closed loop, real-time, and two-level hierarchy (Benders, 1962; Dantzig and Wolfe, 1960) have also been developed to tackle the optimisation issue. In these approaches, a top-level scheduler determines the start and end times of a job, which is subsequently refined by the lower-level scheduling modules. In comparison with these other approaches, the LP and simplex are more tractable in nature and have been used effectively to speed up the process of supplying time-staged deployment, training, and logistical programs in military operations. These approaches

were later generalised as the *mixed integer programming* and the *stochastic programming* paradigm. The complexity of LP was unknown for a long time until in the 1970s Klee and Minty (1972) created an example that showed the exponential time requirement of the simplex method. To control the possible exponential explosion of the search process, the formulation of a correct objective function is a crucial task. Moreover, the correct objective function also provides an exact optimisation criterion, particularly in those situations where different organisational goals are conflicting with each other. Finally, Dantzig (1991) formulated an explicit goal or objective function to show how to guide the search process towards a feasible solution.

Other two popular techniques exist in OR, which tackles the *integer-programming problem: branch-and-bound* (Agin, 1966; Lawler and Wood, 1966) and *Lagrangian relaxation*. The former is an enumerative technique, whereas the latter was devised to remove integer constraints (Shapiro, 1979). Other OR techniques, such as the *Performance Evaluation Review Technique* was originally devised in 1958 for the POLARIS missile program by the Program Evaluation Branch of the Special Projects office of the U.S.Navy, in collaboration with the Lockheed Missile Systems division and the Consultant firm of Booz-Allen & Hamilton (Render and Stair, 1982; Freund, 1979). The *Critical Path Method* technique was devised by M. R. Walker of E. I. Du Pont de Nemours & Co. and J. E. Kelly of Remington Rand, circa 1957 (Render and Stair, 1982; Freund, 1979). In later years, these two techniques have been used extensively in various industries, such as project management, military domain, transportation industry, and supply chain management.

As argued by Ravidran, Philips, and Solberg (1987), all the techniques that have been presented in the preceding paragraphs are suitable only when the problem space is well-defined. Generally speaking, the OR techniques are restricted to rigid and static models with limited expressive power. When implemented in real-life problems their sophisticated mathematical algorithms often result in intractability, mainly because the problem space of the real-life scheduling problems is normally ill-structured. Therefore, the notion of optimality can be troublesome when viewed globally.

In the following section we describe AI approaches to scheduling problem-solving, which attempt to deal with ill-structured, complex real world scheduling problems.

## **2.3 Artificial Intelligence thread in scheduling**

### **2.3.1 Basic concepts in scheduling**

Increasing awareness of the recent developments in the AI community has paved the way for the widespread use of knowledge-based techniques to solve the classic scheduling

problem (Glover, 1986; Grant, 1986). This is due to three main reasons. Firstly, these techniques encompass a rich collection of knowledge representation schemas to deal with the wide range of real-world scheduling problems. Secondly, these techniques provide flexibility, partly due to the use of efficient and flexible problem-solving mechanisms, such as search-based or constraint-based engines, and also because of the use of mixed initiative frameworks. These frameworks enable human experts to represent their problem more systematically. Thirdly, various algorithms have been developed to reflect and deal with the complexities that characterise real-life scheduling problems. Thus, in contrast with OR, the scheduling task in AI has been defined from various perspectives. Below we present some of the influential viewpoints that can be observed in the scheduling literature.

*“Scheduling is the problem of assigning limited resources to tasks over time in order to optimise one or more objectives”* (Pinedo, 2001; Baker, 1974).

Although the above definition emphasises the need to validate a solution schedule against completion and optimisation, it fails to tease out two other equally important validation areas of scheduling: constraint violation and requirement violation.

*“Scheduling selects among the alternative plans and assigns resources and times for each job so that the assignments obey the temporal restrictions of jobs and the capacity limitations of a set of shared resources”* (Fox, 1983).

Fox’s viewpoint takes into consideration the existence of the planning task. While the main function of the planning task is to determine the sequence of actions that need to be performed, the main function of the scheduling task is to allocate these actions over resources and times ranges. This definition also state two types of constraints commonly observed in scheduling - temporal precedence among jobs and limited resource capacity.

*“Scheduling deals with the temporal assignment of jobs to limited resources where a set of constraints has to be regarded”* (Sathi *et al.*, 1985).

*“Scheduling deals with the exact allocation of jobs over time, i.e. finding resources that will process the job and time of processing”* (Brusoni *et al.*, 1996).

Both the above definitions emphasise that the main function of scheduling is to assign jobs to resources by finding a correct time-slot to execute the jobs. While Sathi *et al.* point out the importance of constructing a schedule in accordance with the constraints; Brusoni *et al.* fail to tease out such a type of compliance. Both viewpoints fail to talk about validation of a solution schedule against requirement violation and optimisation.

*“Production scheduling requires allocation of resources (e.g., machines, tools, and human operators) over time to a set of jobs while attending to a variety of constraints and objectives”* (Sadeh, 1994).

Sadeh’s viewpoint defines the scheduling task primarily from the manufacturing perspective. In line with the earlier definitions his notion of scheduling also focuses on validating a solution schedule against completion and constraint violation, but does not consider requirement violation and optimisation.

In sum, we can say that while the above definitions characterise the scheduling task from different perspectives, these proposals are partial in nature as they do not take into consideration the notions of requirement and preference. In our perspective these are important concepts as they help to define the space of valid solutions. To make our viewpoint clearer, in the following section we describe the roles constraints, requirements, and preferences play in constructing a schedule.

### 2.3.2 Constraints, preferences, and requirements

The notion of constraint is central to scheduling. Constraints specify the properties that must not be violated by a solution schedule and therefore they restrict the space of a valid solution. At an abstract level, constraints can be classified into two main categories: *hard constraints* and *soft constraints* (Zweben and Fox, 1994). While hard constraints are prescriptive in nature and therefore they cannot be violated under any circumstances, the soft constraints can be relaxed if required. For instance, in the CIPHER application (cf. Section 8.2), the end-time-compliance constraint states that each work-package must finish exactly on its end time and not prior to it, and a solution schedule that violates this constraint is deemed to be an inconsistent solution. This high-level classification can be further synthesised into the organisational and physical types. Organisational constraints are usually posed in such a way that they increase the profit of an organisation. Typical examples of the organisational constraints include due dates of jobs, work-in-process inventory, resource level maintenance, production levels, shop stability, etc. The physical constraints on the other hand restrict the functionality of a schedule itself. Typical examples of physical constraints include precedence relations among jobs, resource requirements of jobs, resource availability, resource capacity, and so forth (Zweben and Fox, 1994; Fox *et al.*, 1983).

While existing approaches to scheduling treat preferences as a kind of restriction (Smith and Goodwin, 1995; Zweben *et al.*, 1992; Noronha and Sarma, 1991; Fox *et al.*, 1983), we believe that they are rather choice points opted by a scheduler. They are more akin to a

knowledge-level notion, than to the physical demands of a task, to be satisfied by a schedule. Preferences originate from the different alternatives that are available while constructing a schedule. The following example will clarify a need to distinguish between constraints and preferences. For instance, in a manufacturing industry there is a machining shop, which only executes the milling operations, then the obvious constraint is that this shop can perform different types of milling operations that are required by the different engineering components, but then a scheduler may prefer to opt for 'milling-machine-A' or 'milling-machine-B' in order to perform certain type of milling operation even when both the machines are deemed to have a similar functionality. Because preferences are human-specific decisions, they usually affect the cost of a schedule. For these reasons in our scheduling framework we do not involve soft constraints and we treat them as preferences (cf. Section 5.2.6).

Requirements usually specify the properties that a solution schedule must satisfy to become a feasible solution and usually the source of the requirements is a customer specification. While constructing a schedule it is important to keep a clean distinction between the notions of constraint and requirement which essentially are of different in nature. The following example will clarify the necessity. For instance, if a customer requires a machining component in which a hole needs to be drilled of diameter 10 and an upper and a lower tolerance must be 0.5 ( $10^{+0.5}$ ). In order to schedule this job first a scheduler has to satisfy a customer requirement according to which a job needs to be scheduled on the available drilling machines such that a hole can be drilled in a machining component. Having satisfied this requirement, a scheduler must not violate an upper and lower tolerance constraint of a hole of 0.5, and therefore a job must be scheduled only on those drilling machines that can maintain this constraint. Liebowitz and Potter (1995) represent a systematic categorisation of the requirements in terms of general requirements, resource requirements, activity requirements, scheduling capability requirements, rescheduling capability requirements, and output requirements.

The notions of constraint, requirement, and preference make possible to evaluate the performance of a schedule either from a job-based (such as tardiness or due date) perspective or a factory-based (such as throughput and utilisation) one. Figure 2.2 depicts a taxonomy of these notions.

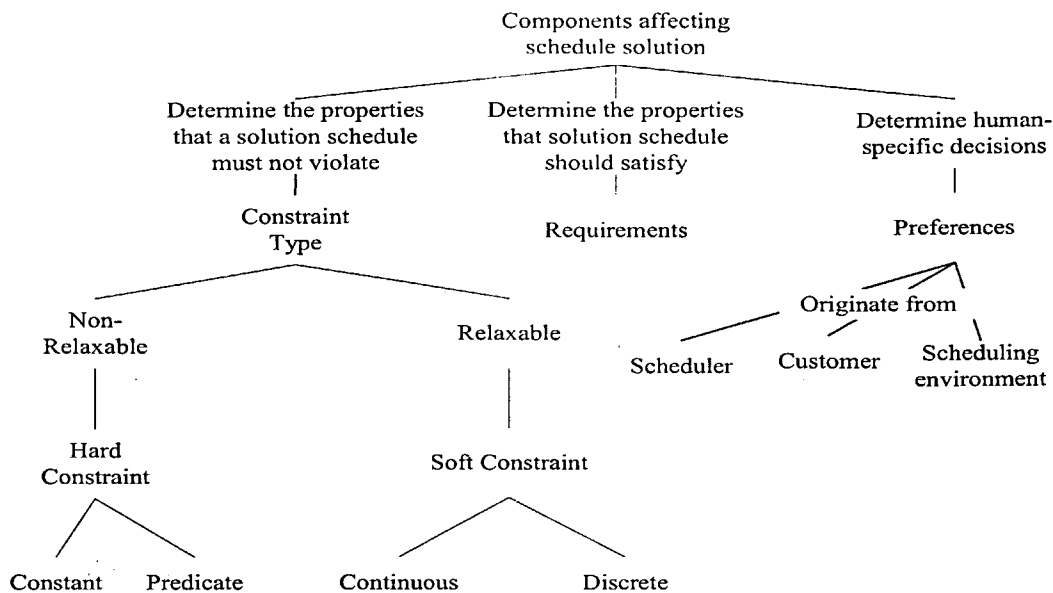


Figure 2.2. Taxonomy of constraint, requirement and preference.

In the following section we will discuss another central concept in scheduling, that of a time element.

### 2.3.3 Theory of time

Time is the crucial factor which distinguishes the scheduling task from the assignment problems. As described in Section 2.2, time is always considered to be discrete in scheduling. Generally speaking, time can be represented in terms of the following elements: time point, time interval, duration, calendar date, and so forth. Over the years, a variety of formal models have been developed to represent time, and they are summarised in Rescher and Urquhart (1971) in the field of philosophy.

A time point represents an instance of time over a time line (Allen, 1983). A time interval or time range represents an amount of time that is elapsed between any two time points. Allen (1983) defines a time range as follows:

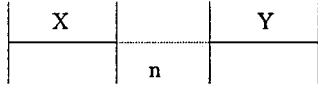
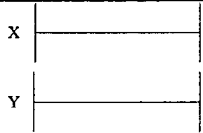
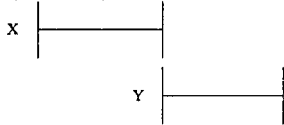
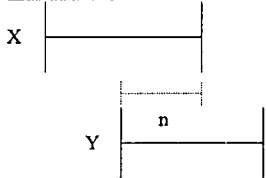
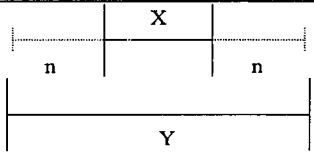
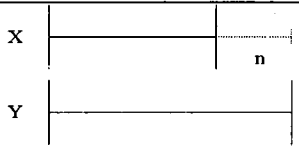
*“A period of time that elapsed between start of an event and end of an event, and start of an event precedes end of an event”* (Allen, 1983).

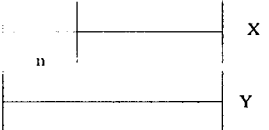
The notion of a time range in scheduling can be used to represent a schedule horizon and a time window within which a job needs to be accomplished. A schedule horizon can be represented in terms of a start and an end time, whereas a job time window can have a more fine-grained representation in terms of the earliest and the latest start and end times. Allen (1983) proposes 13 possible relationships between time events. However, as argued by Zhou and Fikes (2000), Allen’s formalism to represent a time point does not provide good enough granularity. In their framework, a time point is represented at different levels of granularity, such as year, month, week, day, hour, minute, second, etc. They also claim

the following advantages over Allen’s work - a) their framework provides a clean distinction between open and close intervals; b) the level of granularity considered to represent a time point facilitates more precise reasoning; c) their framework takes into account the classes needed to represent calendar months, calendar days, and weekdays.

While constructing a constraint-directed reasoning shell for Operations Management named LOGOS, Meng and Sullivan (1991) have observed that Allen’s temporal relations are under-constrained and that a user might have additional knowledge about time events. For instance, instead of simply stating “event  $a_i$  BEFORE event  $a_j$ ”, a user may like to tighten such a condition by stating “event  $a_i$  BEFORE event  $a_j$  by 3 hours”. Therefore they extend Allen’s temporal relations by imposing a numeric constraint. Table 2.1 provides a comparison between Allen’s temporal relations and Meng and Sullivan’s extended relations. Meng and Sullivan’s numerical constraint is represented by ‘n’ in Table 2.1.

Table 2.1. Comparative analysis between Allen and Meng and Sullivan’s time intervals.

Allen’s relations	Pictorial depiction	Meng and Sullivan’s relations	Translated constraints
X before Y		X before Y (by n)	$x\text{-end} < y\text{-begin}$ or $x\text{-end} = y\text{-begin} + n$
X equals Y		X equals Y	$x\text{-begin} = y\text{-begin}$ $x\text{-end} = y\text{-end}$
X meets Y		X meets Y	$x\text{-end} = y\text{-begin}$
X overlaps Y		X overlaps Y (lag n)	$x\text{-begin} > y\text{-begin}$ or $x\text{-begin} = y\text{-begin} + n$ or $x\text{-end} = y\text{-begin} + n$
X during Y		X during Y <sup>(delay)</sup> <sub>(lag)</sub>	$x\text{-begin} < y\text{-begin}$ or $x\text{-begin} = y\text{-begin} + n$ $x\text{-end} > y\text{-end}$ or $x\text{-end} = y\text{-begin} + n$
X starts Y		X starts Y (by n)	$x\text{-begin} = y\text{-begin}$ $x\text{-end} < y\text{-end}$ or $x\text{-end} + n = y\text{-end}$

X finishes Y		X finishes Y (delay n)	$x\text{-begin} > y\text{-begin}$ or $x\text{-begin} = y\text{-begin} + n$ $x\text{-end} = y\text{-end}$
--------------	--	------------------------	--

These relations among time events can be particularly useful in scheduling to impose the temporal constraints among the time ranges of jobs. In the following section, we describe different models that have been developed to characterise the scheduling task.

## 2.4 Techniques in artificial intelligence

Starting from the early 80s various techniques have evolved which fall under a general category of AI and more specifically into expert systems, KBSs, and several search-based approaches. Based on these techniques several intelligent scheduling systems were developed in 80s and 90s and a detailed review of these intelligent scheduling systems can be found in section 2.5. Here, we discuss the techniques which can be used to tackle the problem.

### 2.4.1 Constraint-based scheduling

Traditionally large numbers of AI problems have been seen as a special case of constraint satisfaction. As described in section 2.3.2, the notion of a constraint is crucial to any scheduling problem. They usually limit the space of a valid solution. Therefore, the constraint-based approaches become quickly popular to tackle the scheduling problem types (Beck and Fox, 1998; Dhar and Ranganathan, 1990; Fox, 1983; Petrie *et al.*, 1989; Prosser, 1989). The constraint-based problem can be formulated by the *constraint-formalism* (Fox, 1983) and *general constraint programming*. The constraint formalism determines how the different constraints in a problem can be represented, while general constraint programming actually solves a problem. The most widely used technique for general constraint programming is the *constraint-satisfaction problem*. Generally understood, the constraint satisfaction problem can be described as follows (Tsang, 1993). There exists a set of variables (jobs), a finite and discrete set of domains (resources and time slots) for a variable, and a set of constraints. The constraints are defined over a subset of a original set of variables, which restricts the combination of values that the variables in a subset can take. The goal is to find one assignment of value to each variable such that a set of constraints are satisfied. The common formulation of the constraint satisfaction problem can be found in (Freeman-Benson, Maloney, and Borning, 1990; Ricci, 1990; Navinchandra and Marks, 1987). The constraint-satisfaction problem usually deals with unary or binary constraints and most common formulation of binary constraints can be depicted by a constraint graph, whereby each node represents a variable and each arc

between two nodes represents a constraint imposed on variables by the end points of the arc. In contrast with LP in which variables can only take the numerical form, in the constraint-satisfaction problem variables can take numerative form as well, such as milling-job-a, milling-job-b, etc. Because the domains of variables in constraint satisfaction are finite, various *lookahead techniques* (Haralick and Elliot, 1980) have been developed to improve the efficiency by exploring the features of the constraint satisfaction problem. Gaschnig (1979) and Gaschnig (1993) developed *intelligent backtracking algorithms* to analyse dead-ends to backtrack toward culprits. Traditionally most of the research in constraint satisfaction has mainly been concentrated on the complete search methods; however, some of the techniques, such as *forward checking* and *fail first* (Haralick and Elliot, 1980) have proved to be efficient in the scheduling domain. The constraint-directed search explores the problem space based on relationships, dependencies, and limitations among the variables. The system stops when a first valid solution (a solution that satisfies all the constraints) is found. Commonly found procedures for the constraint-directed search are *Generate & Test* or *Backtracking strategy* without constraint propagation. Kumar (1992) provides an excellent tutorial-based review of the various algorithms that have been developed for the constraint-satisfaction problem, whereas a generic framework for the constraint-directed search and scheduling is discussed by Beck and Fox (1998). Finally, various intelligent scheduling systems, such as ISIS (Fox, 1983), OPIS (Ow *et al.*, 1988; Smith *et al.*, 1990), SONIA (Collinot *et al.*, 1988), DAS (Burke and Prossor, 1994) constructed by means of the constraint-based approach.

#### 2.4.2 Distributed AI: agents

The research in distributed AI has begun to overcome the limited competence and problem-solving ability exhibited by the single expert systems developed in the 80s. Parunak *et al.* (1985) developed a distributed scheduling approach based on the well-known 'divide-and-conquer' strategy. By using this strategy a problem can be decomposed at various levels and various KBSs co-operate to solve a problem (Zhang and Zhang, 1995). An 'agent' is one of the central notions in distributed AI. An agent is a piece of software that asynchronously co-operates with other agents (Jennings and Woolridge, 1998). Each agent can be seen as a complete KBS in itself. The set of agents may be heterogeneous in terms of their knowledge, goal, languages, algorithms and a multi-agent system can be constructed by selecting and integrating agents with different specifications.

The scheduling task usually comprises the following two types of agents - a task agent and a resource agent. The former agent can be responsible for allocating tasks over resources, whereas a resource agent can be represented in terms of a single resource or a

class of resources to execute the task agents. During problem-solving a task agent sends its request to a resource agent along with the set of operations a resource agent needs to perform. Having received such request a resource agent generates a new schedule to accomplish the assignment. To accomplish the assignment of tasks these agents can be represented by adopting centralised or decentralised architectures. However, there is a debate in the scheduling domain to determine the suitability of centralised or decentralised approaches mainly due to the lack of support for coordination mechanism. Some of the well known scheduling systems constructed by means of distributed AI techniques are OPIS (Ow *et al.*, 1988), SONIA (Collinot *et al.*, 1988), YAMS (Parunak *et al.*, 1985).

### 2.4.3 Artificial neural network

Artificial neural networks try to mimic the learning and prediction ability of a human being. They can be distinguished based on network topology, node characteristics, and training or learning rules. The three-layer, feed-forward neural network is the most simplistic model of artificial neural network. It consists of input layer, hidden layer, and output layer. The *supervised learning* neural network uses historical data to capture the relations between input and output layers. Back-propagation (Rumelhart *et al.*, 1986) is the most popular strategy that subscribes to the gradient-descent technique in the feed-forward network. Rabelo (1990) was the first to apply back-propagation neural nets to solve the JSS problem. Rabelo's JSS problem consisted of different job types and has shown various arrival patterns, alternative process plans, precedence relations, and batch sizes. Another type of model for constructing the artificial neural network is the Relaxation Model. It is defined in terms of the energy functions and there is a pre-assembled system that relaxes from input to output along a predefined energy contour. Hopfield Neural Network (Hopfield and Tank, 1985) is the famous example that subscribes to the Relaxation Model. Initially this model was used to solve some classic textbook scheduling problems by (Foo and Takefuji, 1988), whereas 2-dimensional Hopfield network was used to solve 4-job, 3-machine problems and 10-job, 10-machine problems (Zhou *et al.*, 1990). Finally, the extended version of 2-dimensional Hopfield network is the 3-dimensional network (Lo and Bavarian, 1991), which can be used to represents jobs, resources, and time ranges.

Because the scheduling problem usually consists of several variables, which need to be taken into consideration when generating a schedule, both the aforementioned techniques have suffered due to their computational inefficiency and frequent generation of infeasible solutions. Therefore, they have shown limited applicability to solve real-world applications.

#### 2.4.4 Neighbourhood search methods

Neighbourhood search is an efficient method. When combined with other heuristics it offers good chances of improving the existing solution. These methods usually start by introducing a little perturbation in a complete solution, where a complete solution can be obtained by any greedy search or heuristics, and then, this technique keeps on perturbing a complete solution, until an improvement is achieved in the objective function. In the following section we discuss three such methods: tabu search, simulated annealing, and genetic algorithm.

##### 2.4.4.1 Tabu search

Glover (1989, 1990) has introduced the basic idea of the tabu search. It explores the search space of all feasible schedules by the sequence of moves. The tabu search moves from one schedule to another by evaluating all the candidate solutions, and it chooses the best available candidate. The moves that can potentially get stuck in local optima and hence result in a cycle are classified as tabu moves, i.e. they are forbidden. All such moves are compiled into what is referred to as the *tabu list*. This list is built from the history of previous moves. The tabu moves direct a search to leave the area which contains an old solution and this freeing of search provides ‘strategic freeing’, which can be achieved by a short term memory function.

Tabu search has been applied to solve the JSS and flow shop scheduling problem by Nowicki and Smutnicki (1996) and Vaessens (1995) also showed that tabu search methods are better compared with other neighbourhood search methods, such as simulated annealing, and neural networks. Watson *et al.* (2003) represents a first attempt in the field to quantitatively model the cost of tabu search for any NP-complete problem and particularly for the JSS problem.

##### 2.4.4.2 Simulated annealing

Simulated annealing is an extension of the hill climbing search that tries to escape local minima. Kirkpatrick *et al.* (1983) have proposed this method which is a very general optimization method that stochastically simulates the slow cooling of a physical system. The main concept of simulated annealing is to initialise a temperature as a predetermined starting value and which reduces gradually according to a cooling schedule and to 0 eventually. If the temperature is set a higher value then there are higher chances that inferior moves will be accepted. This method has a cost function, say  $H$  (i.e., a Hamiltonian) which associates a cost to a state in a system, say a temperature ‘ $T$ ’, and there are various ways by which a state of a system can be changed. A current state within a thermodynamic system is analogous to a current solution and its energy equation is

similar to an objective function. Finally, a ground state is equivalent to the global optima. A global temperature 'T' is lowered as the iteration progresses. Similar to the hill climbing method, this method also starts a search from a state that may be generated randomly and in each cycle a random neighbour is examined. If a randomly examined neighbour is better than a current one then it is made a current one; otherwise, a neighbour is accepted only under a probability, which is related to the lowering of the temperature. Based on this analogy Kirkpatrick *et al.* generates a new schedule randomly by sampling the probability distribution of a system. In past, simulated annealing technique was applied to solve the JSS problem. Vakharia and Chang (1990) developed a scheduling system for the manufacturing cells and the resource-constrained scheduling problem was tackled by Jeffcoat and Bulfin (1993).

#### 2.4.4.3 Genetic algorithm

The Genetic Algorithm (GA) (Goldberg, 1989) search method is based on Darwinian natural selection and mutation in the biological systems. It is an optimisation methodology that encodes parallel search with the process of attempting coarse-grained hill climbing. At an abstract level, GA requires the following five components:

1. A fixed length string of symbols for encoding a problem;
2. Evaluation function that could rate for each solution. A typical evaluation function in scheduling can be minimisation of cost or maximisation of resource utilisation, etc;
3. A way to initialise the population of solutions;
4. Genetic operators can be applied on the parent in order to alter their genetic composition. Typical genetic operators are crossover (which randomly selects a segment between parents), mutation (a modified gene), and other domain specific operators;
5. Finally, a parameter setting for an algorithm and the operators, etc.

In the past, several JSS systems were developed by using the GA technique (Davis, 1985; Goldberg and Lingle, 1985; Starkweather *et al.*, 1993). Usually, in JSS problems a GA with blind recombination operators was utilised. Much emphasis was also kept on the relative ordering schema, cycles, and edges in the offspring that could give rise to differences in the blind recombination operators. In contrast with earlier approaches, Bagchi *et al.* (1991) argued that the nature of an evaluation function can be augmented by using the problem-specific knowledge in order to gain more effective results. Uckun *et al.* (1993) have pointed out that the approach adopted by Bagchi *et al.* produces better results only in longer terms as compared with a simpler GA enhanced with local Hill climbing

operator. Starkweather *et al.* (1993) were the first to tackle a dual-criteria scheduling problem by using GA in a real production facility. Their evaluation function primarily aimed at reducing the average inventory level in the plant along with the minimisation of average waiting time of an order. More recently, Burke and Smith (2000) have proposed a hybrid method that combines tabu search, simulated annealing, and GA for the planned maintenance of the national grid.

2.4.4.4 Fuzzy logic

This technique is useful for solving scheduling problems which have uncertain processing times, constraints, and set-up times. The fuzzy set logic theory has been used to develop hybrid scheduling systems. Also, by using a concept of *interval of confidence*, different types of uncertainties can be represented more efficiently. However, these techniques are usually combined with other methodologies, such as search procedure, constraint-based approach, etc. Slany (1994) criticised the straight-forward methods from mathematics that have been adopted to develop fuzzy set logic and he introduced a method called fuzzy constraint relaxation. This method was later integrated with the knowledge-based scheduling in a steel manufacturing plant (Dorn and Slany, 1994). Krucky (1994) focused on a problem to minimise the setup times of the medium-to-high product mix production line whereas Tsujimura *et al.* (1993) developed a hybrid system that could model the processing times of the flow shop scheduling problem.

Table 2.2 shows a comparative analysis of the techniques discussed in this chapter.

Table 2.2. Comparative analysis of different techniques in terms of their usability.

Name of the technique	Overall usability aspect of a technique	Technique-specific remarks
Linear programming	<ul style="list-style-type: none"><li>○ It can be used for the optimisation with linear function</li><li>○ Intractable</li></ul>	<ul style="list-style-type: none"><li>○ A problem must be specified in terms of the conjunctive set of equalities</li></ul>
Distributed AI	<ul style="list-style-type: none"><li>○ Global optimisation</li><li>○ Local perturbation is allowed</li><li>○ Continuous communication among agents is required in order to avoid the global effects made by the local scheduling decisions</li></ul>	<ul style="list-style-type: none"><li>○ User needs to determine about centralised or decentralised representation</li></ul>
Neighbourhood search	<ul style="list-style-type: none"><li>○ Useful for both constraint satisfaction and optimisation</li></ul>	<ul style="list-style-type: none"><li>○ In simulated annealing the neighbourhood function</li></ul>
Tabu search		

Simulated annealing	<p>when near-optimal solutions can be accepted</p> <ul style="list-style-type: none"><li>○ Reflects flexibility in terms of the computation time</li><li>○ Simulated annealing and tabu search tries to escape from the local optima</li></ul>	<p>is crucial to escape from the local optima</p> <ul style="list-style-type: none"><li>○ Determination of a rate at which schedule cools down is crucial</li><li>○ The effectiveness of the tabu search mainly depends on a strategy used for tabu-list manipulation</li></ul>
Neural Network and Genetic Algorithm (GA)	<ul style="list-style-type: none"><li>○ Both the techniques are useful for finding feasible or near-optimal solutions</li><li>○ GA can be implemented for parallel implementation and therefore useful in the real-time applications</li></ul>	<ul style="list-style-type: none"><li>○ In Neural Network determining the network set-up and updating is crucial to gain effectiveness</li><li>○ In Neural Network customisable and specialised networks can be expensive to build</li></ul>
Fuzzy logic	<ul style="list-style-type: none"><li>○ It can be useful in applications where uncertainty is high</li><li>○ It can be used to construct the hybrid systems</li></ul>	<ul style="list-style-type: none"><li>○ The rules used to combine conjunctive or disjunctive clauses can be arbitrary</li><li>○ Rules usually give same importance to all the factors</li></ul>

2.5 Intelligent scheduling systems

The scheduling task did not receive serious attention from AI researchers, until in the 1980s’ Fox *et al.* started developing the first intelligent scheduling system called ISIS. In later years, several intelligent scheduling systems were developed to tackle the scheduling problem from several domains. Here, we review the most influential intelligent scheduling systems developed over the last 20-year period: ISIS (Fox, 1983), OPIS (Ow *et al.*, 1988; Smith *et al.*, 1990), SONIA (Collinot *et al.*, 1988), YAMS (Parunak *et al.*, 1985), FlyPast (Mott *et al.*, 1988), S2 (Elleby *et al.*, 1988), DAS (Burke and Prossor, 1994), REDS (Hadavi *et al.*, 1992), and BATTLE (Slagle and Hamburger, 1985).

Although, these intelligent scheduling systems exploited various techniques in AI successfully, they were hardwired in nature due to their domain specificity. In our review we explicitly focus on the following three aspects: the domain tackled by the system, the

problem-solving technique adopted by the system, and the schedule validation area covered by the system.

### 2.5.1 ISIS

ISIS was developed by Mark S. Fox and his group at the Carnegie Mellon University for a turbine component plant. ISIS formalises the scheduling task by subscribing to JSS. ISIS uses a frame-based knowledge representation approach and its problem-solving strategy is based on the constraint-based beam search. Each constraint in ISIS is represented as a distinct unit and it has an associated utility factor. A utility factor measures an extent to which a particular constraint contributes in validating a solution schedule.

ISIS decomposes the scheduling task into the following four-tiers: lot selection, capacity analysis, resource analysis, and reservation selection. The lot selection module selects a candidate lot for its release on the shop floor. At the second level, the capacity analysis module determines the start time of each job within a selected lot. The actual scheduling operation is performed at the resource analysis level, where each candidate resource is checked against its availability and capacity such that it can accomplish an execution of assigned job. Finally, at the reservation selection level, a candidate resource is reserved for assigning a job, such that the *work in process* inventory is reduced. ISIS subscribes to the job-based perspective to produce a schedule with minimal job lateness. ISIS deploys forward and backward scheduling strategies to assign the jobs. Initially, all the jobs are assigned by applying a forward scheduling strategy (i.e., starting from their start time) and then a backward scheduling strategy (i.e., starting from their due dates) is applied to assign outstanding jobs. During problem-solving the evaluation function is constructed dynamically within each state of a search space. As mentioned earlier, each constraint contributes both to the importance and utility factor in constructing a final schedule and a solution schedule is validated for the job lateness and the constraint satisfaction. However, ISIS fails to reason about requirement violation and optimisation aspects of scheduling.

### 2.5.2 OPIS

OPportunistic Intelligent Scheduler (OPIS) is a successor of ISIS, but in contrast with ISIS it is a reactive scheduling system. OPIS implements the blackboard architecture (Corkill, 1991; Nii, 1986), which is based on a multiple perspective assignment strategy. OPIS was designed to tackle the scheduling problem in a manufacturing domain. The scheduling is performed both on a job-based and a resource-based perspective (Brusoni *et al.*, 1996). OPIS is the first scheduling system that realises the existence of and deals with the bottleneck resources during schedule construction. Before problem-solving the bottleneck areas are detected, by analysing the candidate job and the state of the shop-floor, and jobs

are assigned from a resource-based perspective. These resources are checked against the following criteria: capacity, machine set-up requirements, and batching constraints. Initially, the assignment process anchors a search around the bottleneck areas and once the bottleneck resources are identified then a search is guided on a job-based perspective. Figure 2.3 depicts the blackboard architecture of OPIS.

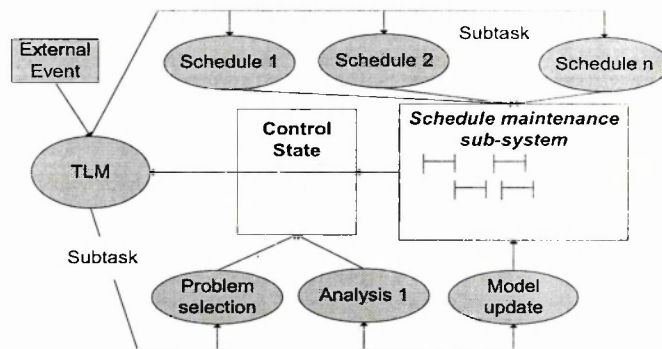


Figure 2.3. Architecture of OPIS (Smith, 1994).

OPIS implements an incremental and opportunistic strategy to solve the scheduling problem, which is the main philosophy of the blackboard architecture. The knowledge applied to solve the scheduling task is distributed across independent knowledge sources (KSs). The application of KSs is determined dynamically and opportunistically as the problem-solving evolves. The KSs executes a system within the blackboard architecture and as a new piece of knowledge becomes available it is augmented in the existing system. The top-level manager (TLM) component is responsible for coordinating different scheduling events. The predictive component of OPIS is more deterministic in nature as compared to the reactive component, because it has more static data available about the problem for its execution. A constraint violation within the reactive component is resolved by using one of the following revision strategies: order-scheduler, resource-scheduler, right-shifter, left-shifter, and demand-swapper. Because OPIS takes into account the notion of a constraint, the solution schedule is validated against constraint violation and also optimisation of a set of objectives to reduce the cost of a schedule.

### 2.5.3 SONIA

SONIA is a successor of SOJA (Le Pape and Sauvé, 1985) scheduling system and it is very similar to the OPIS system. It formulates the scheduling problem based on the JSS model. SONIA was designed to tackle the scheduling problem from the manufacturing domain. SONIA integrates both predictive and reactive scheduling components and works on the blackboard architecture. A predictive component of SONIA is similar in spirit to that of SOJA. It comprises of a job selection and an ordering component. A job selection component aims at selecting a job to be scheduled and binds it to the available resources.

The binding process is carried out by subscribing to the *operations resource reliance* heuristic (Sadeh and Fox, 1996). The ordering component consists of an iterative constraint-satisfaction process (Le Pape, 1994), which imposes the temporal constraints on a selected job. If the ordering component encounters a failure, then a system enters into a reactive scheduling phase. The scheduling decisions are made through predictive component and the backtracking takes place via reactive component. The blackboard architecture of SONIA subscribes to the constraint propagation as its main problem-solving strategy. It consists of the following three components: *KSSs*, *blackboard data structure*, and *control cycle*. First, SONIA subscribes to the micro-opportunistic scheduling (Sadeh and Fox, 1996) during which each module make a collection of decisions about whether to assign a complete manufacturing order or a complete set of resources (Smith *et al.*, 1990). Then a problem-solving strategy relies on the macro-opportunistic approach that allows selection and adaptation of the micro opportunistic strategy. A schedule is validated for completion and constraint violation, but it fails to reason about requirement violation and optimisation aspects of scheduling.

#### 2.5.4 YAMS

YAMS (Yet Another Manufacturing Systems) is probably the first intelligent scheduling system that truly exploited the distributed AI in the manufacturing scheduling domain. YAMS subscribes to the *contract net* (Smith, 1980) as its main architecture. In contract net modules, a transfer of control is in a distributed fashion by using a metaphor of negotiation among the agents. The agents in YAMS's contract net are categorised as being *manager*, *bidder*, and *contractor*. Each category of agent plays a specific role in the contract net. Initially, a *manger* agent identifies a work to be done and delegates it among agents through the negotiation process. Different *bidder* agents offer to perform a delegated work and a *contractor* agent is a successful bidder who wins a contract. The communications within different agents take place via message passing such that if there is any potential task to be performed a manager agent makes an announcement and broadcast a task to all other agents. The agents that have a potential to perform the task contact *manager* with a bid message and the highest *bidder* is awarded the contract which then becomes a *contractor*. Similar to the first approach, the *contractor agent* decomposes a task further into smaller subtasks and acts as a *manager*. An agent can both be *contractor* and *manager* at the same time. This style of problem-solving has been referred to as a *fractal* style (Parunak *et al.*, 1985). YAMS model the complete factory as hierarchical work-cells and an individual work-cell within a factory is considered as an agent in that contract net. A node at a particular level of hierarchy indicates a level of granularity associated with that

particular level, whereas a leaf agent corresponds to the discrete resources. Each node has a collection of plans associated with it, which represents its capabilities. A global schedule is initially performed by using an external system that is distributed across net and then a local schedule is devised by using the *turnpike* strategy (McKenzie, 1976). The problem-solving strategy perturbs a local schedule but the ultimate goal is always to return a global schedule whenever possible. The communication among a contract net takes place only between *superior-to-subordinate* and *peer-to-peer* communication is prohibited. A *peer-to-peer* communication is required to propagate the effect of constraints among agents. However, the main problem with this kind of architecture is that the local scheduling decisions can have global consequences that could make a global schedule obsolete. To overcome this problem the contract net is applied only when a problem is already decomposed into the sub-problems. This kind of strategy is referred to as a *functionally accurate cooperative distributive strategy* (Lesser and Corkill, 1981).

### 2.5.5 FlyPast

FlyPast (Mott *et al.*, 1988) is a resource allocation scheduling system that assigns aircrew to aircraft. To handle the dynamic environment in which FlyPast has to operator it subscribes to a reactive scheduling strategy. FlyPast uses constraint-based reasoning along with an *assumption-based truth maintenance system* (ATMS) (De Kleer, 1986) as its problem-solving strategy. Generally understood, a nature of the problem is simplified in FlyPast because the system performs an assignment once the timings of the flights are predetermined and only decision like allocation of the aircraft crew to the aircraft is considered as a constraint. This problem can be treated as one that of constraint satisfaction and therefore the problem is represented by the constraint-graph. FlyPast subscribes to a resource-based scheduling approach. Each node represents a flight and the domain of a node corresponds to the possible aircrew that can be assigned to it. The arcs between any two nodes represent constraints that exist among them. FlyPast problem-solving algorithm subscribes to the forward checking look ahead heuristic (Haralick and Elliot, 1980) to improve the search efficiency. ATMS nodes are generated for the domain reduction and a datum of an ATMS node is a reduction in a domain achieved from forward checking. If the forward checking strategy results in a *total destruction* of a domain, then a 'no good' solution is derived and the search gracefully descends to the dependency-directed backtracking. If no satisfactory result occurs then a 'no good' database is analysed and delivered to user. The user of the system can interact with the system by adding and retracting constraints or by forcing specific allocations. The final solution is validated against the number of constraints that has been satisfied by a schedule.

### 2.5.6 S2

S2 is similar in spirit to that of FlyPast and it is developed to tackle the VLSI wafer fabrication. The problem domain of S2 can be considered similar to JSS. Because of the dynamicity and uncertainty of the domain S2 treat its problem as an open world and implements a reactive scheduling strategy. A reactive scheduling strategy of S2 also helps to address its domain-specific issues directly. The architecture of S2 is composed of the following three modules - *constraint maintenance system* (CMS), *schedule generator*, and *request interpreter*. The CMS module is used to represent the scheduling problem and constraint propagation is performed every time any constraints are imposed or retracted in CMS. Although, a final solution schedule in S2 is validated against constraint satisfaction, it is built on an assumption that no single performance measure can be used to measure a final schedule. The schedule generator then reacts to the addition or retraction of constraints by modifying the existing solution instead of having to schedule from scratch. The entire process of schedule construction, constraint satisfaction, problem modification, and problem-solving is referred to as an *incremental constraint satisfaction*. S2 uses the depth-first search with dependency-directed backtracking and to recover from the dead ends it keeps record of all the dead ends encountered along with their source of inconsistencies. A hard-wired ATMS is implemented and 'no good' database is distributed across the soft-constraints. Similar to FlyPast, S2 first delivers a satisfactory schedule and then allows its users to modify a schedule either by adding or retracting constraints via *request interpreter*. S2 is based on the *funk box* scheduling strategy (Elleby *et al.*, 1988). It assumes that user of a system has a prior knowledge about how a satisfactory schedule looks like, and therefore, user can guide the scheduling procedure towards a *good* schedule.

### 2.5.7 DAS

Distributed Asynchronous Scheduler (DAS) system is developed at the University of Strathclyde to tackle the problem from the manufacturing scheduling domain. It is a reactive scheduling system, which subscribes to a bottom-up approach of a schedule construction. The scheduling problem is distributed among three types of problem-solving agents: S-agents, T-agents, and O-agents. Figure 2.4 depicts the three-tier architecture of DAS.

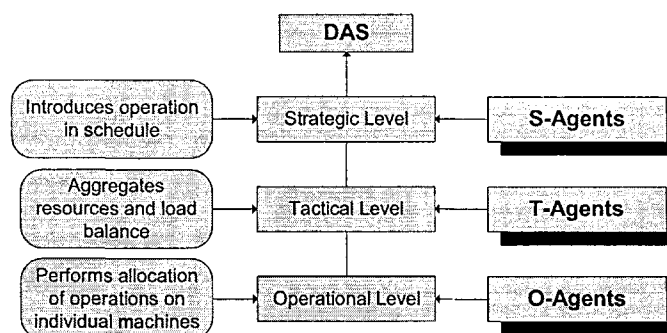


Figure 2.4. Architecture of DAS.

At the *operational level*, the O-agents are responsible for scheduling operations on individual resources. The O-agent uses hybrid algorithm that consists of forward checking (Haralick and Elliot, 1980), shallow learning (Decthter *et al.*, 1990), and dependency-directed backtracking (Stallman and Sussman, 1977). At the *tactical level*, the T-agents are attached with the aggregate resources and the load-balance operations with the subordinate O-agents. Finally, at the *strategic level*, the S-agents are responsible for introducing a work into a schedule. The S-agents have unlimited control over the conflict resolution and they can relax a problem if and when required. All the three agents act asynchronously and constraint propagation takes place through message passing similar to YAMS (cf. Section 2.5.4). DAS can also be seen as one of the functionally accurate communication architecture (Lesser and Corkill, 1981) as that of YAMS. The accuracy within communication architecture is achieved because of the locally accurate decisions made by the O-agents and these local decisions are subsequently distributed among other agents to avoid having global effects.

### 2.5.8 REDS

The Real Time Distributed Scheduling (REDS) is developed by Hadavi *et al.* (1992), which can be seen as a logical successor of DAS. Because REDS is based on the assumption that the scheduling objectives conflict with each other and it is desirable to have a scheduling system that can observe its environment from different perspectives, it has subscribed to a distributed architecture for real time scheduling. REDS is developed for the VLSI fabrication in production scheduling domain. It is based on a philosophy that in scheduling it is not usually clear what exactly needs to be optimised and therefore by distributing the scheduling problem among group of agents it may allow to optimise a problem over various criteria. Most of the architecture of REDS is similar to that of DAS; however, the distributed structure of the scheduling problem among agents is fully exploited in REDS as opposed to DAS. REDS is described to be an attempt to merge both AI and OR techniques in its design. The conceptual architecture of REDS divides the scheduling task into four subtasks: pre-processor, feasibility analysis, detailed scheduler,

and sequencer. The pre-processor module pre-processes new orders arriving in system, the feasibility analysis module is responsible for critical resource scheduling and release control, the detailed scheduler module assures the validity of a schedule, and the sequencer dispatches module that works based on previous module's perspective. REDS performs scheduling both from predictive and reactive perspectives by subscribing to the release control strategy (Hadavi *et al.*, 1992). This strategy helps to reduce the job waiting times, work-in-process, finished goods inventory, and also helps to meet due dates by reducing cycle times. Scheduling in REDS is performed both from job-based and resource-based perspective. In the predictive scheduling, the agents have specific problem-solving capability according to their position in a hierarchy and in the reactive scheduling each agent operates independently by identifying impact of disturbances. The final aim of REDS is to devise a schedule that does not violate any constraints.

### 2.5.9 BATTLE

BATTLE is a decision making expert system for the resource allocation problem in the military domain. Each military weapon in this domain is represented as a resource and each military target as a task. The objective function is an unexpected reduction in the value of targets. BATTLE uses the *computation network*, which is built beforehand to reason with the logical, Bayesian, and expert-defined operators. It is a rule based system, where rules are specified by the domain experts. This network resembles to the *prospector inference network* (Duda *et al.*, 1979), where information is propagated and combined by using Bayes' rule and logic operators. The *computation network* is acyclic which contains a set of nodes and a set of directed links that connects two nodes. Moreover, each node is also associated with two kinds of information: *datum function* and *assignment function*. Each node has a default value associated with it, which is returned if no specific information is supplied. The *assignment function* can be built from the family of evidence functions, which make assignments with a Subjective Bayesian method (Duda *et al.*, 1976). BATTLE also acquires its information in order to reflect a current situation, such as the current situation in the battle field, which is acquired by finding those questions with a high ratio of probable importance to their difficulty. It is referred to as a *merit system* (Slagle *et al.*, 1984).

BATTLE invokes a two phase allocation algorithm. In the first phase *effectiveness* of each weapon against each target is analysed, where effectiveness is measured by the expected proportion of the target that would be destroyed if the weapon were fired at it. The second phase of algorithm uses individual effectiveness from the first phase to evaluate a plan. A good allocation plan is sought by optimising successive weapons.

BATTLE also has the interactive scheduling component that allows its users to enter, augment, and alter data.

With the description of the BATTLE system, we conclude our review of the intelligent scheduling systems. Table 2.3 represents a comparative analysis of these intelligent scheduling systems.

Table 2.3. Comparative analysis of intelligent scheduling systems.

Name of the system	Domain specificity	Problem-solving technique	Scheduling perspective	Scheduling components	Exploits ontologies and problem-solving methods
ISIS	JSS	Constraint-directed search	Job-based	Predictive	No
OPIS	Manufacturing production scheduling	Constraint-based blackboard architecture	Job-based and resource-based	Reactive	No
SONIA	JSS	Iterative constraint-satisfaction blackboard architecture	Job-based	Predictive and reactive	No
YAMS	Manufacturing scheduling	Distributed AI (fractal style)	Not clear	Reactive	No
FlyPast	Air-gate assignment	ATMS + constraint graph	Resource-based	Completely reactive	No
S2	VLSI wafer fabrication	Hardwired ATMS + constraint maintenance system	Not clear	Reactive	No

DAS	Manufacturing scheduling	Distributed AI + constraint maintenance system	Job-based	Reactive	No
REDS	VLSI fabrication production scheduling	Distributed constraint satisfaction with A*	Job-based and Resource-based	Predictive and reactive	No
BATTLE	Military domain	Prospector inference network = subjective Bayesian method and logic operators	Resource-based	Predictive and reactive	No

### 2.5.10 Summary so-far

All the intelligent scheduling systems described in the previous sections failed to gain wider applicability mainly because they were subscribing to a specific scheduling domain. As a result, a new system had to be built from scratch every time the nature of the problem changed. As pointed out by Kruger (1992), such a brittle nature of the system components increases time and cost resources invested in a system construction. As we will discuss in the next chapter the research in the knowledge modelling domain has paved the way for making the system components reusable.

In the following section we will discuss different dispatching rules and heuristics developed to improve the efficiency of the job selection while constructing a schedule.

## 2.6 Dispatching rules and heuristics for the job selection

Constructing a valid schedule in accordance with the different constraints, requirements, and preferences that could emerge in real-world scenarios is a challenging enterprise. In such environments the selection of a correct job is a crucial activity because it improves the efficiency of a schedule construction by reducing unnecessary backtracking. Over the years, various dispatching rules, orders, and heuristics have been developed both in OR and AI, which will be discussed below.

Panwalkar and Iskander (1977) discuss more than one hundred scheduling rules. In their approach the scheduling rules are classified into the following three types – a) simple priority rules, combination of simple priority rules, and weighted priority indexes and b) heuristic scheduling rules. Wu (1987) proposed the three meta-categories in which the dispatching rules can be classified. The first category is based on simple priority rules which make use of the information relating to jobs. This is similar in spirit to the category ‘a’ reported by Panwalkar and Iskander. It can further be subcategorised based on processing time (shortest processing time (SPT)), due date (earliest due date), arrival time (first in first out (FIFO)), and slack (minimum slack). The SPT rules were first studied in detail by Conway *et al.* (1967) and they pointed out that the application of SPT reduces the average mean flow time of jobs. Similar observations were carried out to determine the effect of dispatching rules to optimise the job properties, such as due date and tardiness and the shop properties, such as throughput and utilisation. The second category is based on the combination of rules from the first category. For instance, initially a job selection in this category can be achieved by using FIFO until a queue is 10 jobs long and then SPT can be used to select a job. The last category is referred to as a Weight Priority Indexes, which selects a job by assigning weights to the relative importance of jobs. The relative importance among jobs is usually represented by an objective function. This category is similar to the ‘weighted priority indexes’ category reported by Panwalkar and Iskander.

In comparison with OR, different heuristic commitment strategies have been developed in AI which belong to the constraint satisfaction community. Haralick and Elliot (1980) proposed the *fail-first heuristic* (FFH). According to this heuristic the next best variable (job) is the one which is most likely to fail in a schedule, i.e. most likely to be in dead end. In later years, Freuder (1982) subscribed to the notion of highly constrained variables and proposed the heuristics named *minimal-width-ordering*. This heuristic aims to instantiate variables (jobs) that are highly constrained, in the hope that backtracking will be reduced. For instance, if  $V_A$  constrains  $V_B$  to value X and  $V_C$  to value Y, then this heuristic selects  $V_A$  as a candidate because it would reduce the number of chances in  $V_B$  and  $V_C$  that otherwise create restriction elsewhere. On the other hand, if  $V_C$  is instantiated first then it naturally creates a conflict with  $V_A$ . Dechter and Meiri (1989) have proposed the *dynamic search rearrangement* heuristic and which is similar in spirit to FFH. The dynamic search rearrangement heuristic suggests that if more than one job competing for the same resource, then the best candidate is the one that has least number of resources left for the assignment, and therefore, this job represents the least reliance available. Based on the notion of reliance, Sadeh (1991) have proposed an *operation resource reliance-filtered*

*survivable schedule* (ORR-FSS) heuristic. Generally understood, ORR first identifies the most critical activity as the one that has maximum reliance on the available resources and time ranges for which there is a highest contention and then FSS rates a quality of all the possible start times that can be assigned to the critical activity. Finally, a start time with the highest quality is selected for the assignment. The *Task-Interval-Entropy* heuristic (Caseau and Laburthe, 1995) uses the notion of task intervals as a basis for estimating the resource contention. A similar notion that of a slack was also adopted by Cheng and Smith (1995) whereby they make use of the *precedence constraint-posting* slack along with the constraint-based analysis propagator. Baptiste, Le Pape, and Nuijten (1995) also used the notion of a slack to find a *minimum resource slack* with highest average contention and they have examined the following three heuristics: *choose first*, *choose last*, and *choose dynamically*. Finally, the *minconflicts* (Minton *et al.*, 1992) is a local search heuristic that chooses an activity with the highest violations (i.e., an activity that relies on the most contended for resources and time slots), and for each of its possible start times the number of resulting violations is assessed. Finally, a start time with the least number of resource violations is chosen for assignment. A more detailed analysis of job selection strategies can be found in (Beck and Fox, 1998).

## 2.7 Scheduling in a nutshell

In this chapter we have reviewed and summarised different areas of scheduling research, which have emerged over the years. Based on this review of the field, in a nutshell, we can say that the scheduling task is “*an assignment of time-constrained jobs to time-constrained resources within a pre-defined time framework, which represents the complete time horizon of a schedule. Normally an admissible schedule must not violate any of the constraints imposed on jobs or resources and must satisfy all the input requirements. More in general, the output of the scheduling task is a legal schedule in accordance with a given solution criterion (e.g., complete, admissible, feasible). Preference specific decisions can influence the cost of a schedule*”. Our definition of scheduling is consistent with the earlier definitions discussed in section 2.3, but it also emphasises the need for considering the notions of requirements and preferences to validate a solution schedule along with completion and constraint violation. Moreover, it also emphasises that preferences can affect the cost of a schedule.

Our review began by discussing the different problem types for formulating the scheduling task. Having discussed a formulation of the scheduling task, we reviewed an OR thread in scheduling research, and then we focused our review on scheduling research in AI. Initially, we have seen various viewpoints to define the scheduling task. To this end,

we can say that at a generic level the scheduling task can be characterised by the following eight elements: job, activity, resource, time range, constraint, requirement, preference, and cost. A solution schedule has usually to satisfy a number of conditions, such as completion, constraint violation, requirement violation, and optimisation. As it has been pointed out in section 2.3.3, the notion of a time range can be used to represent a schedule horizon, which subsequently can be specialised to represent the time range of jobs. To highlight the various types of constraints in scheduling, our review presented different scheduling models, based on which we can say that the typical constraints that can be observed in scheduling are precedence constraints, limited capacity of resources, resource requirements of each job, and due dates of a job. In section 2.4, we focused on reviewing different techniques, which can be used to solve the scheduling task. These techniques have varying degrees of knowledge requirements, which affected their usability and implementations to tackle the scheduling task. Then in section 2.5, we reviewed various intelligent scheduling systems, which were constructed by successfully using the different AI techniques. Finally, we reviewed dispatching rules and heuristics developed to select a correct candidate job.

In sum we can say that, the OR approaches to scheduling have certain limitations due to their static formulation and insufficient expressiveness to tackle the complexities from the real-world scheduling problems. Moreover, their primary aim was to achieve an optimal solution schedule, which also was difficult if not impossible in real life. On the other hand, various intelligent scheduling systems have been constructed by using AI techniques. However, the domain specificity of these systems made them hardwired and inflexible in nature, and therefore, these systems had limited reusability. In the knowledge modelling domain system reusability was achieved by constructing the libraries to tackle the generic tasks (Motta, 1999; Valente *et al.*, 1998; Benjamins, 1995; Chandrasekaran, 1990).

In the next chapter, first we will review different components involved in constructing the libraries. Then we focus on reviewing different scheduling libraries and task ontologies that have been constructed. These past efforts allow us to formulate our research basis.

# Chapter 3

## KNOWLEDGE MODELLING APPROACHES TO SCHEDULING

In this chapter, we review the knowledge modelling approaches to scheduling problem-solving. This review is particularly important for us, because it allows us to formulate our research basis.

As described in Chapter 2 (cf. Section 2.5), all the intelligent scheduling systems developed over the last two decades were hardwired and inflexible in nature because they were subscribing to specific scheduling domain. Reusability was the main concern of research in knowledge modelling. Here, the construction of a KBS can be conceived by applying libraries of PSMs (Motta, 1999; Valente *et al.*, 1998; Benjamins, 1995; Chandrasekaran, 1990) to tackle the classes of generic tasks (Chandrasekaran, 1986), such as parametric design, planning, diagnosis, design, etc. Our research subscribes to this stream whereby we aim to construct a generic library of scheduling PSMs. Ontologies and PSMs are the two central components in the library construction process. These two components are reviewed in the following section. Then, in section 3.2 we will review the existing scheduling libraries and in section 3.3 we will review the existing scheduling task ontologies. The review presented in these two sections is particularly important for us because based on these past efforts we formulate our research basis. In section 3.4, we analyse the gaps in the existing approaches to the scheduling library construction and task ontologies. Finally, in section 3.5 we conclude the chapter by indicating what needs to be done in order to bridge the gaps in the existing scheduling libraries.

### 3.1 Ontologies and problem-solving methods

In the 1991, the ARPA knowledge sharing effort (Neches *et al.*, 1991) proposed a novel perspective for knowledge sharing while constructing intelligent systems. Their proposal was as follows:

*“Today’s development process of knowledge-based systems (KBSs) relies on building knowledge-bases from scratch. To avoid the brittle nature of these KBSs, the process of constructing KBSs will begin by assembling and subscribing to the existing reusable components. Therefore, the knowledge engineers can reuse the existing knowledge bases, thus leaving only with the worry of constructing their specialised reasoning patterns embodying in PSMs. This would facilitate constructing expanding and enriched systems much cheaply in terms of time and cost”.*

In compliance with the above proposal, a system's reusability can be augmented by the abstract reusable reasoning patterns underlying a KBS that are usually referred to as a PSM. Libraries of these reusable PSMs can be constructed to tackle different types of generic tasks. A formulation of the Generic Tasks approach (Chandrasekaran, 1986) was particularly instrumental because it highlighted a clear distinction between a task specification (the problem to be solved) and a method (that can be executed to solve a task). Each task can be solved by applying different methods, which can further be decomposed into several (-sub) tasks and (-sub) methods. A knowledge modelling framework provides a methodology to construct a library that systematically organises different building-blocks associated with a library. Some influential examples include, Generic Tasks Structures (Chandrasekaran *et al.*, 1992), Role-Limiting Methods (Marcus, 1988), Protégé-II (Musen *et al.*, 1993), CommonKADS (Schreiber *et al.*, 1994), MIKE (Angele *et al.*, 1998), Components of Expertise (Steels, 1990), EXPECT (Swartout and Gil, 1995), GDM (Terpstra *et al.*, 1993), VITAL (Domingue *et al.*, 1993), and Task-Method-Domain-Application (TMDA) (Motta, 1999).

While constructing a library of reusable components one very important decision needing to be taken by knowledge engineers is how to represent the acquired knowledge within a system. Consistently with Newell's proposal of 'knowledge level hypothesis' and 'principle of rationality' (cf. Section 1.1), and in line with the work by Motta (1999), Steels (1990), and Breuker and Wielinga (1985), among others, knowledge can be systematically represented at the knowledge level independently of its physical realisation in a computational system that enables a scheduling agent to achieve its reasoning functionality. Another aspect of the knowledge modelling paradigm is that the KA process is driven by pre-existing knowledge models, often represented as *ontologies* (Gruber, 1995). In the following section we review the current research on ontologies.

### 3.1.1 Ontologies

Ontologies primarily aim at capturing static domain knowledge. They allow knowledge engineers to represent a commonly agreed conceptualisation of domain knowledge, which can be shared and reused over wider applications and groups. Ontologies are usually organised in terms of *taxonomies*. They consist of the following modelling components: classes, relations, rules, functions, axioms, and instances (Gruber, 1993). The taxonomic organisation of the concepts provides a structure for the inheritance mechanism. Relations represent a specific interaction between different concepts. Functions can be seen as a special case of relations, where the  $n$ -th element of a relationship is unique for its  $n-1$  preceding elements (Gruber, 1993). Axioms are used to express the principles, the rules

that are always true in the universe of discourse (Gruber, 1993). More importantly, axioms determine the competence of ontologies. Finally, instances are used to represent individual elements.

According to Gruber (1993) the notion of an ontology can be defined as follows:

*“An ontology is an explicit specification of a conceptualisation”.*

Although, Gruber’s viewpoint is the most widely referred one, as argued by Guarino (1997) the main problem with this definition is that it relies on the notion of ‘conceptualisation’ which is introduced by Genesereth and Nilsson (1987) to formalise the meanings, whereas in reality the notion of a conceptualisation can only be understood intuitively. Recently, Poli (2002) raises three questions which should be considered while constructing ontologies: “what are the boundaries of ontologies?”. That is, what problems are ontological rather than epistemological, logical, or linguistic, etc. “What are the types of ontologies?”. Poli proposes the following three types: descriptive, formal, and formalised, and each category can further be treated as domain specific and domain independent. Finally, “what is the structure of an ontology?”. Poli suggests that the structure of an ontology can be defined by the *theory of items*. Other definitions of ontologies can be found in (Motta, 1999; van Heijst *et al.*, 1997; Valente and Breuker, 1996; Borst *et al.*, 1995; Guarino and Giaretta, 1995; Neches *et al.*, 1991).

Ontologies can be built by subscribing to the following principles: clarity, coherence, extendibility, minimal ontological commitments (Gruber, 1993), minimisation of the semantic distance between sibling concepts (Arpirez-Vega *et al.*, 1998), and finally ontological distinction (Borgo *et al.*, 1996). These principles are described below.

- **Clarity:** States that the intended meaning should be communicated effectively without any ambiguity by providing appropriate sufficient and necessary conditions;
- **Coherence:** States that the internal consistency must be maintained. At least axioms should maintain logical consistency because they determine the competency of an ontology;
- **Extendibility:** States that ontologies should leave scope open to extend the existing terms such that it does not require much revision of existing definitions;
- **Minimal ontological commitments:** States that as few claims as possible should be made while developing an ontology;
- **Minimisation of semantic distance between sibling concepts:** States that the similar concepts should be grouped together and represented as subclasses of one class and

should be defined by using the same primitives. On the other hand the concepts that are different than each other should appear at a distance in a hierarchy;

- **Ontological distinction:** States that the classes corresponding to different identity criteria must be disjoint.

Different methodologies have been developed to construct ontologies, including: Enterprise Ontology (Uschold and King, 1995), TOVE (**T**Oronto **V**irtual **E**nterprise) methodology (Grüninger and Fox, 1994). Bernaras *et al.* (1996) have presented a methodology in the domain of electrical networks as a part of ESPRIT project named KACTUS. The METHONTOLOGY methodology (Fernandez *et al.*, 1997) enables ontology building at the knowledge-level and their framework is supported by ODE (Blazquez *et al.*, 1998). Finally, the SENSUS methodology (Swartout *et al.*, 1997) arrived a year later than METHONTOLOGY.

Here, we do not discuss in detail the different categories and types of ontologies, but rather concentrate on their classification that is useful to represent the knowledge associated with a generic task during library constructing. A more detailed discussion about ontology classification can be found in (Gomez-Perez and Benjamins, 1999). Ontologies can be classified into the following four broad categories: task ontology, method ontology, domain ontology, and application ontology. These categories are described below.

- **Task Ontology:** Formalises the nature of a generic task by providing different concepts, relations, function, and axioms, which are associated with it ideally application domain independently. Motta (1999) defines a task ontology primarily from the knowledge modelling perspective, whereas Mizoguchi *et al.* (1995) conceptualises a task ontology as a result of their interview system named MULTIS (Tijerino and Mizoguchi, 1993);
- **Method ontology:** Provides the lexicon necessary to specify the problem-solving behaviour of a particular method (Musen *et al.*, 1994; Tijerino and Mizoguchi, 1993), whereas according to Coelho and Lapalme (1996) a method ontology specifies the declarative definition of inferences;
- **Domain ontology:** Represents the knowledge associated with a specific domain, either in a task-specific or a task-independent way. While a task-specific viewpoint of domain ontologies is mono-functional, the task-independent viewpoint is generic as it does not subscribe to any specific task (Motta, 1999). Typical examples of the task-independent domain ontologies are Cyc (Guha and Lenat, 1990), PhysSys ontology (Borst *et al.*,

1995), EngMath (Gruber and Oslen, 1994), time ontology (Fikes and Zhou, 2002), etc.;

- **Application ontology:** Contains a set of vocabulary for conceptualising a particular application (van Heijst *et al.*, 1997; Gennari *et al.*, 1994). Because these ontologies concentrate on a specific application of a task they are non-reusable in nature.

Here, we conclude our discussion about the ontologies and in the following section we provide an overview of the notion of a PSM.

### 3.1.2 Problem-solving methods

A PSM can be used as a model-based template to direct the KA process (van Heijst *et al.*, 1992) and to support robust and maintainable applications by reuse (Motta, 1999; Marcus, 1988). The notion of a PSM is present in all the knowledge modelling frameworks enumerated earlier. PSMs describe the inference process underlying a KBS in an *implementation* and *domain-independent* way (Fensel and Benjamins, 1998). For instance, Clancy (1986) abstracted the problem-solving behaviour exhibited by different rule-based systems into a common and generic inference pattern called “heuristic classification” at the knowledge-level.

PSMs can be classified into the following two categories: *task-specific* and *task-independent*, depending upon whether or not they subscribe to a specific class of generic tasks. McDermott (1988) refers to *task-specific* methods as *strong methods* because they tackle the specific classes of generic tasks. A systematic taxonomic representation of strong methods can be found in Marcus (1988). Task-independent methods on the other hand do not subscribe to any specific class of generic task and therefore are usually referred to as *weak methods*. The term ‘weak’ here indicates that these methods do not exhibit any assumptions about the type of task that can be solved by their application. They rather tackle a problem at a high-level of abstraction, such as search (Newell and Simon, 1976). Fensel and Benjamins (1998) point out that a PSM makes an effective use of the domain knowledge in order to achieve the goal of a task and based on this viewpoint a PSM can be characterised as: 1) the specification of inference actions for solving the goal of a task, 2) the definition of one or more control structures over the actions, and 3) a set of knowledge roles indicating how domain knowledge is used during its execution.

The most influential stream of research in the field of PSMs is the *development of libraries of PSMs* and their *reuse*. Because PSMs are developed to tackle a specific task, it is useful to abstract and formulate them at a generic level such that they can be reused to construct a new PSM quickly. Gomez-Perez and Benjamins (1999) proposes the following

categorisation for classifying libraries of PSMs: generality, formality, granularity, and size. We augment this classification by proposing a new category: '*domain specificity*'. While the *generality* dimension determines whether a library is developed to tackle a specific generic task, the *domain specificity* adds another layer of granularity by pointing out whether an entire library or part of a library is defined to tackle a specific domain. Based on the *domain specificity* dimension one can determine whether a library is reusable within a single domain or the multiple domains of a generic task. For instance, the generality criterion would highlight a library for the scheduling task, while domain specificity would provide a more specific pointer stating that a library is in fact constructed to tackle the production scheduling task, e.g., Hori and Yoshida's (1998) library. Based on the generality dimension one can state that a library has wider reusability whilst the reusability of a library can have reusability within a single domain according to the domain specificity. The categorisation proposed by Gomez-Perez and Benjamins (1999) is described below:

- **Generality:** Determines whether the PSMs are developed to tackle a specific task. The typical examples of task-specific libraries are diagnosis (Benjamins, 1995), parametric design (Motta and Zdrahal, 1996), planning (Valente *et al.*, 1998), assessment (Valente and Löckenhoff, 1993), and so on;
- **Formality:** Classifies a library into informal, formal, and implemented ones. Informal libraries provide a structured textual representation of PSMs (Chandrasekaran, 1990), formal libraries allows the verification of the properties of PSMs (Benjamins and Aben, 1997; ten Teije, 1997; Aben, 1993), and implemented libraries provide operational specification of PSMs (Gennari *et al.*, 1994; Puerta *et al.*, 1992);
- **Granularity:** Determines whether the libraries are developed to tackle a complete task, such as the parametric design library (Motta and Zdrahal, 1998) or the fine-grained parts of the task (Aben, 1993). However, many libraries comprise both the types and the former are built from the latter ones (Motta and Zdrahal, 1998; Barros *et al.*, 1996; Benjamins, 1993; Chandrasekaran, 1990);
- **Size:** Characterises a library based on the number of PSMs included in the library and determines how many types of tasks it tackles. CommonKADS (Breuker and van de Velde, 1994) is the most comprehensive library that tackles the following tasks: diagnosis, prediction of behaviour, assessment, design, planning, assignment and scheduling, and engineering modelling.

Because the ultimate aim of this thesis is to construct a generic library of scheduling PSMs, it is essential to look at the different types of knowledge-intensive PSMs that can be applied to tackle the scheduling task.

### 3.1.3 PSMs for the scheduling task

According to Wielinga and Schreiber (1997) different types of configuration processes, such as assignment, planning, scheduling, configuration, etc. can be treated as synthesis tasks. The configuration process can be defined as *a form of design where a set of predefined components are given and an assembly of selected components is sought that satisfies requirements and obeys a set of constraints* (Mittal and Frayman, 1989). The configuration process often assumes a structure of components, where the components may be objects or processes, symbolic or physical, and the connections that are present among these components. In the same paper, Wielinga and Schreiber have proposed a taxonomic representation of the different knowledge-intensive PSMs that can be applied to tackle the synthesis task. Instances include: Propose and Backtrack (P&B) (Runkel *et al.*, 1996), Propose and Revise (P&R) (Marcus and McDermott, 1989), Propose and Exchange (P&E) (Poeck and Puppe, 1992), Propose and Improve (P&I) (Motta, 1999), Propose and Genetical-Exchange (P&GE) (Poeck and Gappa, 1993), etc. At an abstract level, these PSMs follow the similar philosophy proposed by the Propose-Critique-Modify family of methods (PCM) (Chandrasekaran, 1990). Figure 3.1 depicts the taxonomic representation of PSMs proposed by Wielinga and Schreiber (1997).

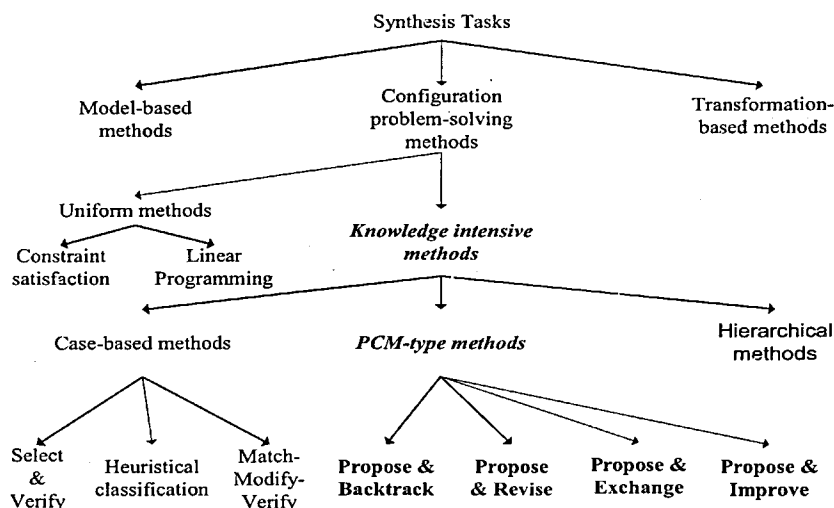


Figure 3.1. Taxonomy of the methods applicable to the synthesis task.

The PSMs depicted in Figure 3.1 can be classified into the knowledge-intensive ones and domain-independent uniform methods like, constraint-satisfaction and LP. We are not interested in the domain-independent methods, because they do not fully exploit domain-specific knowledge during their problem-solving. Constraint-satisfaction has a long history

as a problem-solving technique to tackle the scheduling task (Cesta *et al.*, 1999; Beck *et al.*, 1998; Beck and Fox, 1998; Cheng and Smith, 1995; Dorn and Slany, 1994; Kumar, 1992; Fox and Sadeh, 1990; Fox, 1983). However this uniform approach to modelling fails to provide a fine-grained epistemological framework to analyse the different knowledge-intensive tasks take place in scheduling. It is essentially an implementation technique.

Although, the case-based methods can be used effectively to tackle the other types of tasks, in scheduling they can have limited applicability due to two main reasons. First, they aim to find the best candidate solution from the set of available solutions; however, in real-life dynamic and uncertain scheduling domains this issue can act as a bottleneck, because a scheduler may prefer to rely on the currently available knowledge instead of consulting past cases. Second, case-based methods rely on *blame assignment* to decide which aspect of the stored solutions that has caused the constraint violations (Wielinga and Schreiber, 1997). However, in scheduling it is not always possible to find the exact source of knowledge associated with the culprit decisions, because a combination of several situations may have contributed to devise a poor solution. Therefore, a scheduler may have to retract several inter-linking decisions to reach the source of a conflict. Nevertheless, a scheduling system called CABINS (Miyashita and Sycara, 1994) presents a methodology for learning a control level model for selection of heuristic repair.

During our library construction we are mainly interested at comprising aforementioned knowledge-intensive PSMs, due to two main reasons. First, these methods make extensive use of the domain-specific knowledge during problem-solving. Second, the different phases involved in these methods can complementarily be used both to construct and repair a schedule. In this sense they exhibit the characteristics of both constructive and repair method (Zweben *et al.*, 1993).

In the following section, we review the Propose and Revise method in the context of scheduling. The development and description of other PSMs can be found in Chapter 7, where we engineer them as a part of a library.

### 3.1.3.1 Propose and Revise

The Propose and Revise (P&R) method was originally developed to tackle VT, a system for elevator configuration (Marcus and McDermott, 1989). Because one application could not prove the generic nature of this method, it was modified to tackle the production scheduling problem (Stout *et al.*, 1988). Later it was integrated with the KA tool called SALT (Marcus and McDermott, 1989). This method decomposes the scheduling task into three sub-tasks: 1) propose an extension to a schedule by applying *procedures*, 2) check

the currently extended schedule for constraint violations, and 3) revise a schedule in order to fix the constraint violations by applying *appropriate fix strategy*.

P&R does not rely on the explicit information about all the components and their connections. In the propose phase, the assignment of jobs to resources and time ranges is achieved by applying the procedures. The jobs can be selected based on the domain-specific and search-control knowledge. After an assignment of each job the verification phase is invoked to evaluate the constraint violations. Usually, the constraints are evaluated based on the domain-specific perspective. If constraint violations occur during the propose phase, then the revise phase is introduced to tackle the constraint violations by applying the *fixes*. Because of the antagonistic nature of the constraints in scheduling, the revise phase is invoked only after a complete schedule is devised (Stout *et al.*, 1988). However, Motta and Zdrahal (1998) and Zdrahal and Motta (1995) have proposed the following two strategies to fix constraint violations: *extend-model-then-revise* (EMR) and *complete-model-then-revise* (CMR) in the context of the parametric design. EMR fix the constraint violations as soon as they occur while constructing a solution, whereas the constraint violations are fixed in CMR only when a complete solution is devised. The order over application of different fixes can be determined based on the application specific knowledge. According to Wielinga and Schreiber (1997), the main limitation of the P&R method is that fix application usually relies on the heuristic knowledge and therefore it can be biased towards a specific solution types.

## 3.2 Existing libraries for scheduling

Here, we review the following scheduling libraries that were developed in the past: the production scheduling library (Hori and Yoshida, 1998), CommonKADS library for assignment and scheduling tasks (Sundin, 1994), Le Pape's library of constraint-based scheduling (Le Pape, 1994), and MULTIS-II (Tijerino and Mizoguchi, 1993).

### 3.2.1 Hori and Yoshida's library for production scheduling

Hori and Yoshida (1998) have proposed a domain-specific library for the production scheduling task. The library construction in their approach subscribes to a bottom-up approach, whereby the knowledge requirements of each problem-solver are realised based on the production scheduling domain. The library is organised into three main components - the task level, the problem-solving level, and the domain level. At the task level, the scheduling task is formalised by its task ontology. Because the task ontology component will be discussed separately in section 3.3, here, we describe the other two components.

The problem-solving level consists of different tasks and methods and they are organised according to the data flow (i.e., control knowledge) among them. The control knowledge among different problem-solving inferences is clarified based on a domain model of production scheduling. In our perspective, the main problem of this kind of commitment is that it makes difficult to identify the knowledge requirements of the different problem-solving tasks and methods independent of the domain. Thus, the cost of reuse is very high. In line with the earlier approaches to library construction (Motta and Zdrahal, 1996; Valenete *et al.*, 1998; and Valente and Löckenhoff, 1993), this library provides a clean separation between the problem-solving and domain knowledge. The scheduling engine consists of the two sub-systems: the **dispatch method** and the **assignment method**.

The **dispatching method** provides a high-level control structure that determines the priority of each lot instead of concentrating on the assignment of individual units, which is achieved by the following three methods: **reset**, **removeTop**, and **isEmpty**. First, the **reset** method creates an initial queue of all the lots that are still unassigned in a schedule; then the **removeTop** method returns a lot with the highest priority; and finally the **isEmpty** method is invoked after each cycle to check whether a queue is empty, otherwise it invokes a cycle.

The **assignment method** mainly has to do with the actual scheduling operation at a more fine-grained level by subscribing to the forward scheduling strategy, which helps to prevent units being delayed. This method assigns units to resources by fixing their start and end times. The assignment is accomplished by the following three operations: **reset**, **isDone**, and **doNext**. The **reset** initialises the unit queue, the **isDone** checks whether all the units in a queue are assigned, and otherwise the **doNext** operation executes assignment of a focal unit to the selected resource. The method **checkUnit** is invoked to validate whether the focal unit assignment complies with the constraints, and in case of failing to maintain the constraints, it analyses how to fix them. The constraint violation associated with the assignment of a time range is dealt with the method **modifyTime**. All the outstanding units are then assigned by using the *backward scheduling strategy* based on their due dates, which help to reduce the in-process inventory. Although, this library validates a solution schedule against completion and constraint violation, it fails to reason about two other important schedule validation criteria, requirement violation and optimisation.

### 3.2.2 CommonKADS library

CommonKADS is a comprehensive methodology that supports the construction of libraries of the task specific PSMs. It also tackles assignment and scheduling tasks. However, it only consists of the P&R method (Marcus and McDermott, 1989). The detailed discussion

on the KADS model of the scheduling task can be found in (Balder *et al.*, 1993). The CommonKADS library subscribes to a top-down approach of the library construction, whereby a high level task is decomposed into (-sub) tasks and (-sub) methods. This decomposition structure is similar in spirit to the *generic tasks* proposal (Chandrasekaran, 1986).

The CommonKADS library consists of the following two variants of the P&R method: *simple P&R* and *hierarchical P&R*. The former one is similar to the original description of the P&R method (cf. Section 3.1.3.1). In the *hierarchical P&R*, first high-level units (jobs) are assigned to the high-level resources. In the propose phase first all the unassigned units are sequenced based on a certain ordering criteria, and then the function **select unit** selects a candidate-unit for its assignment. The function **propose-assignment** executes assignment of a selected candidate-unit to resources by two methods: **one-step resource matching** and **step-wise resource matching**. The former method matches the resource requirement of a unit directly against all the resources, whereas the latter method matches the resource requirement against one of the several resources assignable to a unit. The **step-wise resource matching** method is particularly useful when several resources can be assigned to a unit.

The revise phase is invoked if any constraint violation occurs while assigning the units. A flawed set of assignments are revised by the function **modify**. The constraint violations are fixed by using one of the following methods: **re-try-the-last-assignment**, **local-exchange**, **global-exchange**, **generic-fixes**, and **scaling-of-constraints**. The method **re-try-the-last-assignment** falls back to the last consistent assignment of a unit. The **local-exchange** (Poeck and Puppe, 1992) fixes the constraint violations by exchanging the assignment of the units in conflict. Finally, the **global exchange** strategy performs a series of exchanges to fix constraint violations.

Because each method in their library is realised by its direct association with the high-level task, it makes difficult to realise how the existing inferences can be reused or to construct a new PSM quickly. Moreover, because the CommonKADS library consists of the P&R method, the only schedule validation criteria that can be tackled are completion and constraint violation, but it fails to tackle requirement violation and optimisation.

### 3.2.3 Constraint satisfaction approach for resource allocation scheduling

As described earlier, constraint satisfaction has long been used as problem-solving technique for scheduling (Dorndorf *et al.*, 2000; Cesta *et al.*, 1999; Beck *et al.*, 1998; Beck and Fox, 1998; Cheng and Smith, 1995; Dorn and Slany, 1994; Fox and Sadeh, 1990; Fox,

1983). In line with the traditional approaches, ILOG SCHEDULE also subscribes to the constraint satisfaction as its problem-solving strategy to tackle the *resource allocation* problem. ILOG SCHEDULE is built on top of the SOLVER, which is a generic software tool that provides the object-oriented programming environment.

The resource allocation problem exhibits the following characteristics: 1) the availability of each resource varies over time, independently of the availability of other resource-types, 2) the resources can be aggregated into their more abstract forms, and 3) the resource availability at some point in time depends on the availability of other resources at other points in time (Baptiste and Le Pape, 1995). To reconcile these three characteristics of the problem, ILOG SCHEDULE consists of a generic framework of the *resource time-table*. It maintains the information about resource utilisation and resource availability over time. Two types of implementations are proposed for representing the resource time table in ILOG SCHEDULE. The first implementation assumes a discrete representation of the time. It also keeps a history about maintaining the status of a variable at any instance of time over the complete time interval. The second implementation does not make any assumptions about the nature of time, whether discrete or dense, but simply maintains a history about the time when the status of a variable changed from unassigned to assigned one. These two implementations are referred to as the *discrete array* and the *sequential table* respectively. The assignment of an activity to resources is achieved by the second mechanism and by complying with a generic *disjunctive constraint*. A disjunctive constraint restricts the overlapping of incompatible activities in time. Finally, the third mechanism called *edge-finding* takes as an input arbitrary tuples of activities and makes sure that a certain activity must precede or succeed other activity. This mechanism is also responsible for assigning a precise earliest and latest start and end times to jobs and activities.

The nature of the resource allocation problem in ILOG SCHEDULE is formalised by the object model. The resources and activities are two central concepts in the resource allocation problem. The notion of a schedule in the library is represented by the class CtSchedule. It is represented by a set of activities, a set of resources, and a time interval covered by a schedule. The notion of resource is formalised by defining the class CtResource. The resource capacity constraint is formulated by the following two methods: the *cumulative formulation* and the *disjunctive formulation*. The class CtActivity defines the notion of an activity. It is represented by a start time, end time, and duration. Finally, the duration of an activity is directly proportional to the amount of resource capacity that is consumed by each activity during its execution.

### 3.2.4 MULTIS-II

MULTIS-II proposes a conceptual programming environment called CLEPE, which allow its end-users to incorporate their problem-solving at the conceptual level with the problem-solving inferences of a system. The authors claim the following three advantages of the CLEPE environment: 1) it provides human-friendly primitives whereby the end-users can quickly describe their problem-solving process, 2) a task ontology within their system simulates a problem-solving process at an abstract level, and 3) it provides the environment for an ontology author to construct their own ontology. The CLEPE environment is built as the Generic Process Network (GPN), where each node in GPN represents a *generic process* and a *link* between any two generic processes represent a control flow among them. In the following paragraph we describe the problem-solving process adopted in MULTIS-II.

The domain specific problem-solving conceptualisation of the end users is translated at the task and problem-solving level by using the '*task-domain binding mechanisms*'. These mechanisms act as glue to integrate the domain specific concepts of end users with the task specific ones. Having translated the domain specific conceptualisations, the problem-solving process of MULTIS-II is invoked. Below we describe the problem-solving process of MULTIS-II during any  $k^{\text{th}}$  iteration. The complete process is iterated until a complete schedule is devised.

1. Initially, all the unassigned jobs in a schedule are collected and grouped together based on their similarity measure;
2. All the resources are then classified by using the task **classify-schedule-resource**;
3. The jobs collected in step 1 are then sorted in a particular order by complying with the domain-specific specifications, such as a job with earliest due-date, etc.;
4. All the classified resources are then sequenced in a certain order. MULTIS-II classify resources based on their incremental order from least to highest quantity of load handled by the resources;
5. The task **take-job** makes a certain job from the list of unassigned jobs as a focal job. A certain job is made a focal candidate according to the domain-specific knowledge (cf. point 3). Having made a certain job as a focal job, the task **pickup-job** is invoked to pickup a focal job;
6. Once a focal job is selected then all the resources that can be assigned to a focal job are classified in an incremental order that represents the amount of load that can be handled by a resource. A candidate resource from the list of classified resources is

selected by the task **pickup-RSC**. This task subscribes to a default criterion that selects a resource with the lowest load. The task **pickup-RSC** uses a causal relation named **select** to select a resource;

7. Finally, the task **assign-RCP-to-RSC** takes as an input a focal job and a selected resource and executes their assignment. The relation called **consists-of** represents a permanent assignment of a job to its resource.

Once an assignment of the currently selected job is completed then the load over other resources is updated by the task **update-load**. The complete procedure is iterated in  $(k+1)^{th}$  iteration until all the jobs in a schedule are assigned to devise a complete schedule. Each newly assigned job is appended to the **assignment-set**, which represents a final solution.

In the following section we review the existing scheduling task ontologies.

### 3.3 Existing scheduling task ontologies

While reviewing the existing scheduling task ontologies we primarily focus on analysing the following characteristics: different concepts used to formalise the scheduling task, domain-specificity or independence of these task ontologies, and their subscription towards particular problem-solving technique. Here, we do not review generic enterprise resource ontology (Fadel *et al.*, 1994) and common ontology defined for the DARPA/Rome planning and scheduling initiative (Allen and Lehrer, 1992), because instead of providing a complete task ontological framework these initiatives rather concentrate on formalising specific scheduling components. Our review consists of the job assignment task ontology (Hori *et al.*, 1995; Hama *et al.*, 1992a; Hama *et al.*, 1992b), the MULTIS ontology (Mizoguchi *et al.*, 1995), and the OZONE ontology (Smith and Becker, 1997).

#### 3.3.1 Job assignment task ontology

The job assignment task ontology was developed since 1987 under the project named CAKE (Hori *et al.*, 1994). Currently, it is residing on the Ontolingua server<sup>1</sup> (Farquhar *et al.*, 1997). The job assignment task can be defined as “*assigning all given jobs to the available resources within the time range, while satisfying various constraints*”. In their framework the scheduling task can be realised by the following four components: job, resource, time-range, and constraint. The concepts *job*, *resource*, and *time-range* are represented as *entities* and the notions of *assignment* and *constraints* as *relations*.

---

<sup>1</sup> Please refer to the following URL: <http://www-ksl.stanford.edu/knowledge-sharing/ontologies/html/job-assignment-task/index.html>

The concept *job* denotes an entity that can be assigned over resources and has a specific time slot. It is further classified into *unitary-job* and *aggregated-job*. The class *job* has the following types of subclasses: **temporally-fixed-job-class**, **fixed-length-job-class**, and **unit-length-job-class**. The function **job.time-range** is defined which allows to assign a time range to a job. To determine the temporal order among any two jobs the following relations are defined among any two jobs: **unordered set**, **ordered set**, and **interval**. Table 3.1 report attributes of the class *job* and its subclasses.

Table 3.1. Attributes of the class job and its subclasses.

Class	Attributes
Job	job-name, job-type, job-length, job-time-range
Unitary-job	Job-member-of, job-assignable-resource
Aggregated-job	job-member-set

The concept *resource* defines an entity to which job can be assigned. The class *resource* is further specialised into two subclasses: *unitary-resource* and *aggregated-resource*. The former type of resource cannot be divided into smaller units, whereas the latter type of resource denotes a group of unitary and aggregated resources. Table 3.2 show the attributes of the class *resource* and its subclasses.

Table 3.2. Attributes of the class resource and its subclasses.

Class	Attribute
Resource	Resource-name, resource-type
Unitary-resource	Resource-member-of, assignable-resource-of
Aggregate-resource	Resource-member-set

The concept *time-range* denotes a certain period of time to which a job can be assigned. A job time range is represented in terms of *start-time*, *end-time*, and unit of time. This task ontology subscribes to Meng and Sullivan (1991) (cf. Section 2.3.3) to represent the temporal relations among jobs.

The class *assignment* represents an assignment of a *unitary-job* to resources and time ranges, which is achieved by the function **assigned-resource** and **assigned-time-range** respectively.

In their framework, the constraints are classified into *direct constraint* and *indirect constraint*. The former type of constraint is classified based on whether it restricts an assignment of a resource or a time range. The latter type of constraint is classified according to its characteristics in scheduling, such as job-specific and timetable-specific.

This task ontology validates a solution schedule only against the completion and constraint violation, but it fails to take into account requirement violation and optimisation.

We see this task ontology as a straightforward framework that models the job assignment task. It formalises the job assignment task according to a job-based perspective (Fox, 1983). This task ontology has limited reusability because it subscribes to the job assignment task. Moreover, the level of granularity of different concepts in this task ontology is very coarse and their fine-grained analysis is typically missing. Therefore, it does not provide an adequate framework to characterise the scheduling task precisely. Because the modelling definitions of the concepts do not include any slots, it is difficult to realise how application specification knowledge can be acquired by filling the slots of these definitions. Finally, a schedule validation criterion fails to reason about requirement violation and optimisation issues because the relevant concepts, such as requirements and cost that are required to validate them are missing from their framework.

3.3.2 OZONE

The OZONE ontology was developed at the Carnegie Mellon University for configuring a constraint-based scheduling system. It is a result of a prior experience in building planning and scheduling systems from the different domains, such as manufacturing production scheduling (Smith, 1994), space mission planning (Muscettola *et al.*, 1992), military evacuation, and aero-medical evacuation (re)-planning (Lessila *et al.*, 1996). OZONE can be seen as a meta-model that defines the scheduling task as a process of feasibly synchronising the use of *resources* by *activities* to satisfy their *demands* over time. The OZONE ontology presumes the underlying constraint-directed search architecture (Lessila *et al.*, 1996; Smith, 1994) and provides the necessary base concepts.

The OZONE ontology subscribes to the job-based scheduling perspective (Fox, 1983). It formalises the scheduling task based on the following concepts: demand, product, activity, resource, and constraint. Each concept is represented through the inclusion of *properties* and *capabilities*. While the former corresponds to the *attributes* of concepts and are further classified into *static* and *dynamic* properties, the latter one represents a problem-solving behaviour of the concepts. Table 3.3 define important concepts along with their properties.

Table 3.3. Main components in the OZONE ontology.

Components	Definition	Properties
<b>Demands</b>	It is a request for goods, services, or products	Product, release or due date, temporal relations, priority, and activities
<b>Products</b>	It is goods or services provided	Activities and resources

	by the system of interest	
<b>Resources</b>	It is defined as an entity that supports execution of activities	Resource capacity and resource availability
<b>Activities</b>	It represents a process executed over a certain time interval	Start and end time, assigned resources, duration, resource-requirements, a set of temporal relations, demand, status
<b>Constraints</b>	Constraint restricts the assignment of activities	

The resource is a central concept in the OZONE ontology. It is further classified into **capacitated-resources** and **discrete-state-resources**. The former type is further divided into: **reusable-resource** and **consumable-resource** depending upon whether their capacity can be consumed by the activities. Based on the resource capacity, the resources are categorised into the physical structure, such as **atomic-resource** and **aggregate-resource**. The **atomic-resource** is divided into **unit-capacity-resource**, **batch-capacity**, and the **aggregate-resource** is classified into **homogeneous-resource-pool**, **simple-capacity-pool**, **structure-capacity-pool**, and **heterogeneous-resource-pool**. One of the main limitations of this ontology is that the similar type of analysis like that of the concept resource is missing to conceptualise other concepts.

The notion of constraint is classified into *hard constraint* and *soft constraint* depending upon whether it can be violated during schedule construction. Designation of the soft constraints is accompanied by a specification of objective or preference (e.g., relaxation of a due date).

This ontology subscribes to Allen's (1983) temporal relations to represent the temporal relations among any two activities. The OZONE ontology provides only completion and constraint violation validation criteria to evaluate a solution schedule. However, because the concepts, such as requirement and cost are completely missing from their framework they do not hold any accountability to validate a solution schedule against requirement violation or optimisation.

### 3.3.3 MULTIS

The MULTIS task ontology was developed within MULTIS project at the Osaka University since 1987 by the group of Riichiro Mizoguchi. The MULTIS task ontology was developed through a *task analysis interview* system for the general class of scheduling problem. In their framework the notion of task ontology can be conceived in two ways: 1) task-subtask decomposition along with task categorisation and 2) an ontology to specify

the problem solving process. MULTIS is a “generic vocabulary” that consists of *generic nouns*, *generic verbs*, and *generic adjectives* along with other task related concepts. The *generic process* is a combination of verbs and nouns that occur in scheduling. More than one generic process creates a network of processes referred to as GPN, which is a knowledge-level representation of the scheduling task. The MULTIS task ontology consists of the following four concepts:

- **Generic nouns:** Represent the objects that are necessary in the problem-solving process, such as schedule recipient (e.g., job, order, etc.), schedule resource (e.g., line, machine, etc.), and schedule representation.
- **Generic verbs:** Represent the primitive actions that are executed during problem-solving. Typical examples of the generic verbs include assign, classify, select, pick up, relax, and neglect.
- **Generic adjectives:** Modifies the pre-existing status of different objects in a schedule, such as assigned job, unassigned job, and the last (job).
- **Others:** These are the words which are specific for evaluating the scheduling task, e.g., strong constraint, weak constraint, constraint predicates and attributes.

MULTIS characterises the scheduling task based on the following four basic concepts: *schedule recipient* (RCP) (e.g., a job or an order), *schedule resource* (RSC), *time-slot*, and *constraint*. RCP is a meta-level concept that denotes an entity that can be assigned to RSC and time slot. Different relations are defined over RCPs, such as assigned/unassigned RCP, previous, last, and next. These relations are useful to determine the status of a RCP while constructing a schedule, i.e. whether a RCP is assigned (unassigned) or to represent the temporal relations among them. The class *RCP-GRP* group different RCPs together based on their similarity measure. RSC indicates the entity on which RCP can be assigned, (e.g. a machine) for its accomplishment. RSCs of similar functionality are grouped together into the class *RSC-GRP*. The notion of a time-slot indicates a place where RCP can be assigned to RSC for its execution. Two relations are defined namely *time available* and *assigned time* which represents the status of a time-slot indicating whether it is available or assigned to RCP. The assignment of RCP to RSC and time-slot can be restricted by the constraints, which are classified into strong constraints (hard) and weak constraints (soft). MULTIS solution criterion tries to optimise the priority of RCPs in a schedule. A solution schedule is represented in terms of *generic noun* named *schedule*, which represents an assignment of RSP to RSC within a specific *time-slot*. The function named **assign-RSC\_schedule\_RCP**

executes an assignment of RCP to RSC. Finally, a solution schedule in the MULTIS ontology is validated for the completion and constraint satisfaction.

The MULTIS ontology also provides vocabulary to characterise the problem-solving behaviour of the scheduling task. For instance, the task **classify-schedule-resource** classifies RSC according to its type, the task **sequence-schedule-recipient** sequence RCPs according to their *earliest start time* or *earliest due date*, the task **pickup-RCP** selects a candidate RCP, and finally the **select-RSC** selects a RSC for executing RCP.

In sum, this task ontology characterises the scheduling task at generic level without subscribing to any particular application domain of scheduling. However, MULTIS provides only a partial characterisation of the scheduling task, mainly because concepts such as activities, requirements, and cost are missing in their proposal. As a result, MULTIS framework fails to deal with requirement violations and optimisation issues. More importantly, because MULTIS framework comprises a vocabulary both for a task specification and for describing problem-solving, it blurs a clean distinction between the task ontology to formalise a generic task and method ontology to characterise its problem-solving behaviour. Consistently with other approaches to knowledge modelling, such as KADS (Breuker and Wielinga, 1985), Generic Tasks (Chandrasekaran, 1986), or Components of Expertise (Steels, 1990), we believe that maintaining a clean separation between task characterisation and problem-solving model facilitates the reusability of these components.

### 3.3.4 Summary so-far

Having discussed different proposals to formalise the scheduling task, in Table 3.4 we provide a comparative analysis of all task ontologies that are discussed. The comparison is performed on the following four dimensions: first, we compare these task ontologies based on the different components involved in them. The '√' sign in Table 3.4 indicate whether a specific concept is present in a task ontology while formalising the scheduling task; otherwise, it is indicated by the 'X' sign. Second, the domain specificity column indicates if a task ontology subscribes to a specific domain of scheduling. Third, the problem-solving specificity column indicates whether a task ontology subscribes to any problem-solving technique that can be used to solve the scheduling task. Finally, the schedule validity column indicates the different types of schedule tasks that can be validated by a task ontology. The abbreviations in Table 3.4 represent the following components: J (job), Jt (job-type), A (activity), At (activity-type), R (resource), Rt (resource-type), C (constraint), R (requirement), P (preference), Cs (cost).

Table 3.4. Comparison between different task ontologies.

Task ontology	Components in the task ontology										Domain specificity	Problem-solving specificity	Schedule validity
	J	Jt	A	A t	R	Rt	C	R	P	C s			
Job Assignment	√	√	X	X	√	√	√	X	X	X	Job assignment task	Non generic	Complete and consistent
OZONE	X	X	√	X	√	X	√	X	√	X	Resource allocation	Assumes constraint-based framework of scheduling	Complete and consistent
MULTIS	√	X	X	X	√	X	√	X	√	X	No	Non generic	Complete and consistent

The literature review presented in the previous two sections highlighted the strengths and weaknesses in the existing scheduling libraries and task ontologies. Based on this insight of the field, in the following section we establish our research objectives by analysing the weaknesses in the existing approaches.

3.4 Legacy of the literature review: gap analysis

The limitations that will be discussed in the following subsections will allow us to form the basis of our research. Our aim will be to tackle these limitations by developing a generic library of scheduling PSMs.

3.4.1 Limitations in the existing scheduling libraries

The limitations that we observed in the existing scheduling libraries can be classified into the following four categories: 1) partial coverage of knowledge-intensive PSMs, 2) domain specificity, 3) partial coverage to validate different areas of the scheduling task, and 4) unsuitability for KA.

3.4.1.1 Partial coverage of knowledge-intensive methods

The existing scheduling libraries fail to provide a comprehensive coverage to the different knowledge-intensive methods enumerated in section 3.1.3. For instance, CommonKADS (Sundin, 1994) is the only library that comprises of the P&R method, but it fails to provide any accountability for the other methods such as P&B (Runkel *et al.*, 1994), P&E (Poeck and Puppe, 1992), P&I (Motta, 1999), etc. Other libraries in the field (Hori and Yoshida,

1998; Le Pape, 1994; Tijerino and Mizoguchi, 1993) fail to provide any coverage to these knowledge-intensive PSMs. Because these PSMs make heavy use of the knowledge during their construction, the knowledge roles<sup>2</sup> associated with these libraries can be realised by the domain specific knowledge and it could facilitate KA. More importantly, different phases involved in these PSMs can cover and reason about different types of scheduling tasks, such as completion, constraint and requirement violation, and optimisation.

#### 3.4.1.2 *Domain specificity*

Some of the existing scheduling libraries (Hori and Yoshida, 1998 and Le Pape, 1994) tackle the scheduling task in terms of a specific domain, which limits their reusability. For instance, Hori and Yoshida's library tackles the production scheduling task while Le Pape's library deals with the resource allocation problem. Hence, these libraries cannot be reused over wider domains. The domain specific nature of a system also affects maintenance. As it has been pointed by (Tu *et al.*, 1995), in the lifecycle of a system the task requirements and the available knowledge are likely to change over time, and therefore the maintenance of a monolithic system is difficult. Ideally, we would like to construct a library whose components can provide a wide 'horizontal cover' for the different scheduling domains.

#### 3.4.1.3 *Partial coverage to validate different areas of the scheduling task*

As described earlier in section 3.4.1.1, because all the existing scheduling libraries provide a partial coverage to the knowledge-intensive PSMs, they cannot cover and reason about all the validation areas crucial to scheduling. For instance, the problem-solvers from all the existing libraries primarily focus on validating the scheduling task against completion and constraint violation, but they do not provide any mechanism for dealing with requirement violation and optimisation issues.

#### 3.4.1.4 *Unsuitability for KA*

Some of the existing proposals (Le Papa, 1994) subscribe to a specific problem-solving technique, such as constraint satisfaction and therefore are unsuitable for KA. The constraint satisfaction approach to problem-solving mainly focuses on developing sophisticated but domain independent algorithms that could solve a problem quickly. However, this domain independence makes it difficult to realise what roles the domain

---

<sup>2</sup> In compliance with the Generic Tasks approach (Chandrasekaran, 1986) a top-level task (which in our case is the scheduling task) can be decomposed into a small number of sub-tasks and sub-methods can be proposed to achieve these tasks. These tasks require the application domain specific static and dynamic knowledge for their execution. These knowledge pieces essentially represent the abstract names of data objects that represent the role of these objects in the reasoning steps.

knowledge play while executing the inference actions of PSMs. The knowledge roles for instance can efficiently be used to achieve the goals of tasks through the application of the domain knowledge (Fensel and Straatman, 1998). A PSM that makes effective use of the domain-knowledge can be used to achieve a crucial role in KA. Finally, the constraint satisfaction approach of problem solving does not provide a fine-grained analysis about the different knowledge-intensive tasks occur in scheduling. It is essentially an implementation technique.

In the following section we highlight the limitations of the existing task ontologies.

### 3.4.2 Limitations in the existing task ontologies of scheduling

In our viewpoint the scheduling task ontologies discussed in Section 3.3 fail to provide comprehensive results due to the following reasons: 1) insufficient degree of formalism, 2) domain specificity, 3) commitment to specific problem-solving technique, 4) incomplete characterisation of the scheduling task, and 5) incomplete validation criteria for the scheduling task.

#### 3.4.2.1 Insufficient degree of formalisation

The definitions of the important concepts in existing task ontologies (Hama *et al.*, 1992a, b; Mizoguchi *et al.*, 1995; Smith and Becker, 1997) do not provide the required *level of detail* and *formalism* to conceptualise the scheduling task. More importantly, the properties of these concepts are represented often at a coarse-grained level. For instance, in the satellite scheduling application (cf. Section 8.1) each satellite (i.e., jobs) has a specific requirement for the antennas (i.e., resources) on which they can be assigned to ensure earth-satellite communication activity, each satellite also has a specific time range within which these communication activities need to be completed, and a duration of these communication activities. And it is difficult to realise how this application-specific knowledge can be acquired if the class properties are coarse-grained in nature. To clarify this point, we will take a knowledge modelling definition of the concept *job* and *resource* from the job assignment task ontology<sup>3</sup>, which is the Ontolingua specification (Farquhar *et al.*, 1997).

---

<sup>3</sup> These definitions are taken from the following URL: <http://www-ksl.stanford.edu/knowledge-sharing/ontologies/html/job-assignment-task/job-assignment-task.lisp.html>

```
(define-class JOB (?job)
  :def (source ?job))

(define-class RESOURCE (?resource)
  :def (target ?resource))
```

It can be realised from the above definitions that such a type of conceptualisation does not provide enough expressiveness to capture a particular viewpoint over scheduling precisely. Moreover, because these definitions do not have any slots associated with them to represent the properties of the concepts, it becomes difficult to acquire the application specific knowledge by filling the slots of these definitions.

#### 3.4.2.2 Domain specificity

As it can be observed from Table 3.4, some of the scheduling task ontologies (Hama *et al.*, 1992a, b; Smith and Becker, 1997) subscribe to the specific scheduling domains. A domain specificity of these task ontologies restricts their reusability in a single domain. Therefore, new task ontological model has to be built from scratch every time the domain changes.

#### 3.4.2.3 Commitment to specific problem-solving technique

Some of the existing task ontologies (Smith and Becker, 1997) assume the existence of a particular problem-solving technique while characterising the scheduling task. For instance, the OZONE framework assumes an underlying constraint-directed search as its problem-solving technique (cf. Table 3.4). The disadvantage of subscribing to a particular problem-solving technique is that while characterising the scheduling task the important conceptual distinctions are not considered, if they are not directly supported by the problem-solving environment. In line with the structured approaches to knowledge modelling, such as CommonKADS (Schreiber *et al.*, 1994), TMDA (Motta, 1999), etc., our aim is to provide a clean separation between the task analysis and problem-solving phases.

#### 3.4.2.4 Incomplete characterisation of the scheduling task

As it can be observed from Table 3.4, the existing approaches (Hama *et al.*, 1992a, b; Mizoguchi *et al.*, 1995; Smith and Becker, 1997) fail to provide a comprehensive analysis of all the important concepts, such as activity, requirement, preference, cots, etc. necessary to characterise the scheduling task. In some cases, when they take into account most commonly observed concepts in scheduling, such as job, resource, etc., then their representation into specific forms, such as job-type, resource-type, etc. is typically missing. As a result, such task ontological frameworks fail to capture the scheduling task by teasing out important conceptual distinctions exists in different scheduling environments. For instance, in a manufacturing environment, the notion of a machining operation can be represented by using a concept *job*, but a more specific type of machining operation, such

as drilling machining can only be conceptualised if the concept like job-type is available in a task ontology.

#### 3.4.2.5 *Incomplete validation criteria for the scheduling task*

The solution criteria of all the existing task ontologies validate a solution schedule only against completion and constraint violation (cf. Table 3.4), but they fail to deal with requirement violations or optimisation issues. These are important notions which provide a richer and more exhaustive evaluation basis for a schedule to become a valid solution.

### 3.5 What needs to be done?

To overcome the limitations exhibited by the existing reusable library components of scheduling (cf. Section 3.4.1 and 3.4.2), in our approach we aim to construct a task-specific but application domain independent library of scheduling PSMs. Consistently with the earlier approaches to the library construction, such as parametric design (Motta, 1999), diagnosis (Benjamins, 1995), CommonKADS (Breuker and Van de Velde, 1994), which subscribe to the knowledge modelling framework, our library will be organised according to a knowledge modelling framework. In particular, the TMDA framework (Motta, 1999) will allow us to organise our library systematically in terms of task component, method component, domain component, and application component. In compliance with this organisation, we first construct a generic scheduling task ontology that aims at overcoming the limitations observed in the existing task ontologies (cf. Section 3.4.2). Then we develop a generic problem-solving model of scheduling that provides a high-level abstraction of all the knowledge-intensive tasks and methods necessary to construct more complete problem solvers for scheduling. These high-level tasks and methods will be reused to engineer more specialised PSMs. Our aim is to provide a comprehensive coverage to all the PSMs that are enumerated in Section 3.1.3. Finally to confirm its generic nature, our library will be validated on scheduling applications from different domains.

In the following chapter, we provide a detailed discussion about our library architecture.

# Chapter 4

## ARCHITECTURE OF THE SCHEDULING LIBRARY

This chapter presents the architecture of our library, which is constructed by instantiating the TMDA (Motta, 1999) knowledge modelling framework. This approach enables us to explicitly specify the principles and assumptions underlying our library (van Heijst, 1995), which provides both analytical and engineering foundations for scheduling. Analytically, it exhibits a nice integration of various techniques that have been developed in scheduling research and also provides an insight into the various components necessary to scheduling systems. From the engineering perspective, our library offers support for the rapid construction of scheduling applications in different domains.

The content of the chapter is organised as follows. In the following section we provide a brief overview of the research issues which the library aims to address. Then in section 4.2, we describe our rationale for subscribing to the TMDA knowledge modelling framework. In section 4.3, we discuss the different components of our library: the task, method, domain, and application level. In section 4.4, we characterise scheduling in terms of search problem-solving. In section 4.5, we introduce the OCML (Motta, 1999) language, which will be used as our knowledge modelling language to specify the library. Finally, in section 4.6 we summarise the main points from this chapter.

### 4.1 Statement of the research objectives

Here, we state the key objectives that our library is designed to achieve to overcome the limitations observed in existing approaches to the construction of reusable components for scheduling problem-solving (cf. Sections 3.4.1 and 3.4.2).

- The ultimate aim of our thesis is to construct a task specific and domain independent library of scheduling PSMs. Because our library is domain independent, it not only overcomes the inflexibility associated with the existing domain specific approaches (Hori and Yoshida, 1998; Le Pape, 1994) but is also easier to maintain (Tu *et al.*, 1995);
- To overcome the limitations pointed out in section 3.4.2 in the existing scheduling task ontologies (Hama *et al.*, 1992a, b; Mizoguchi *et al.*, 1995; Smith and Becker, 1997) our task ontology obeys the following criteria: i) it is reusable across scheduling domains and independent of any problem-solving technique that can be used to tackle the scheduling task; ii) it provides a detailed specification of all the components that are

essential to formalise the scheduling task; and iii) it provides a comprehensive set of notions to be able to characterise the different types of schedules;

- At the method level, in line with several earlier proposals (Motta, 1999; Musen *et al.*, 1994; Runkel and Birmingham, 1993; Chandrasekaran *et al.*, 1992; Wielinga *et al.*, 1992; Steels, 1990) our aim is to construct a reasoning component of a library, whereby first a generic model of scheduling problem-solving will be constructed. A generic model of scheduling problem-solving abstracts from the various specific techniques and provides a detailed breakdown of the various tasks and methods carried out in scheduling problem-solving. Our aim is to re-engineer several knowledge-intensive PSMs, such as Propose & Improve (Motta, 1999), Propose & Backtrack (Runkel *et al.*, 1994), Propose & Revise (Marcus and McDermott, 1989), Propose & Exchange (Poeck and Puppe, 1992), Propose & Genetical-Exchange (Poeck and Gappa, 1993) simply by reusing and specialising the small-grained tasks and methods defined in generic model of scheduling problem-solving. Because our library aims at providing a comprehensive coverage of different scheduling PSMs, it can cover and reason about all the validation areas crucial to scheduling, such as completion, constraint violation, requirement violation, and optimisation;
- We also aim to facilitate the KA in a way similar to that provided by role-limiting methods (McDermott, 1988). However, we aim to overcome their restrictive nature (Musen, 1992) by providing a flexible and comprehensive framework for assembling scheduling systems from reusable components.

## 4.2 Rationale for using the TMDA framework

As pointed out in Chapter 3 (cf. Section 3.1), various knowledge modelling frameworks, such as Generic Tasks Structures (Chandrasekaran *et al.*, 1992), Role-Limiting Methods (Marcus, 1988), Protégé-II (Musen, *et al.*, 1993), CommonKADS (Wielinga *et al.*, 1992; Schreiber *et al.*, 1994), MIKE (Angele *et al.*, 1998), Components of Expertise (Steels, 1990), EXPECT (Swartout and Gil, 1995), GDM (Terpstra *et al.*, 1993), VITAL (Domingue *et al.*, 1993), and Task-Method-Domain-Application (TMDA) (Motta, 1999) have been proposed to provide a structured organisation for a library of problem-solving components. For instance, the CommonKADS framework (Wielinga *et al.*, 1992) proposes the following three epistemological categories: task knowledge, inference knowledge, and domain knowledge, Components of Expertise (Steels, 1990) distinguishes between application task, information sources, and problem-solving methods while Protégé-II (Musen *et al.*, 1993) considers task knowledge, method knowledge, domain knowledge, and application knowledge.

In line with the earlier approaches to knowledge modelling, such as CommonKADS and Components of Expertise, the TMDA framework introduces a clean separation between task knowledge, method knowledge, and domain knowledge. However it then extends this partition by introducing an '*application*' component. The *application component* provides a systematic separation between a mapping knowledge and application-specific knowledge. The former is used to interpret a task and method components with multi-dimensional domain models. The need for the mapping knowledge is associated with the domain independence of PSMs. In other words, if there is a mismatch between a domain model and the knowledge requirements of a PSM then it is bridged by defining appropriate *mapping mechanisms* (Gennari *et al.*, 1994).

Similarly with the CommonKADS and Components of Expertise knowledge modelling frameworks, RLM (Marcus, 1988) is one of the influential approaches for constructing generic models. RLM not only facilitates knowledge acquisition, knowledge representation, and efficient inference, but it also provides a clean separation among them. RLM requires a certain domain model organisation and then it provide the control mechanisms, which can be applied on the domain model for making efficient inferences. Generally understood, RLM makes the following three basic claims: 1) there exists a family of tasks that can be solved by the application of methods and the control knowledge of these methods can be abstracted independent of their family specific characteristics, 2) if any of the methods whose control knowledge is task-independent then such methods can make effective use of the task-specific knowledge to achieve the identification, selection, and implementation of actions, and 3) the reasoning efficiency of PSMs can be improved by separating the representation of the control regimes from the task knowledge.

RLM subscribes to a problem-solving as a basis for identifying, selecting, and implementing the sequences of actions to accomplish a task from a specific domain. A selected method provides a way to identify the selection of a potential action at any given time and it also provides one or more mechanisms to select among the candidate actions. The control knowledge in RLM consists of an algorithm which specifies when to use a particular type of knowledge, and it emphasises that the knowledge that method requires for selecting among candidate actions is not the control knowledge. Therefore, it maintains a clean separation between the control knowledge and the problem-solving knowledge.

In summary, RLM is important historically because it was one the approaches, which for the first time implemented Clancey's *role differentiation principle* (Clancey, 1992). However, in RLM a notion of a PSM is "*an algorithm which determines how domain-specific knowledge is used for solving problems*", and therefore, a PSM is a hardwired one

which provides a specific functionality that reduces their reusability. In contrast with this, in TMDA a PSM defines a class of problem-solvers that can be used to solve a task and it exploits the unique functionalities of these problem-solvers. Finally, one of the shortcomings of the RLM approach is that it failed to exploit the notion of application ontology in order to formalise the application specific knowledge.

The notion of *application ontology* was first introduced in the Protégé-II framework (Gennari *et al.*, 1994). However, as it has been pointed out by Guarino (1997), in the Protégé-II approach the application ontology is mainly used to construct a tool that can be used to instantiate the application knowledge base and in the work by van Heijst *et al.* (1997) the use of application ontology is realised to update the ontology library. In both approaches the construction of application ontology is a creative process with very limited support for explaining what concern the actual content of application ontology itself. In contrast with both the earlier proposals, the notion of application ontology in TMDA provides a systematic organisation of the concepts that may be present in an application knowledge base. Figure 4.1 shows the organisation of our library in terms of the TMDA framework.

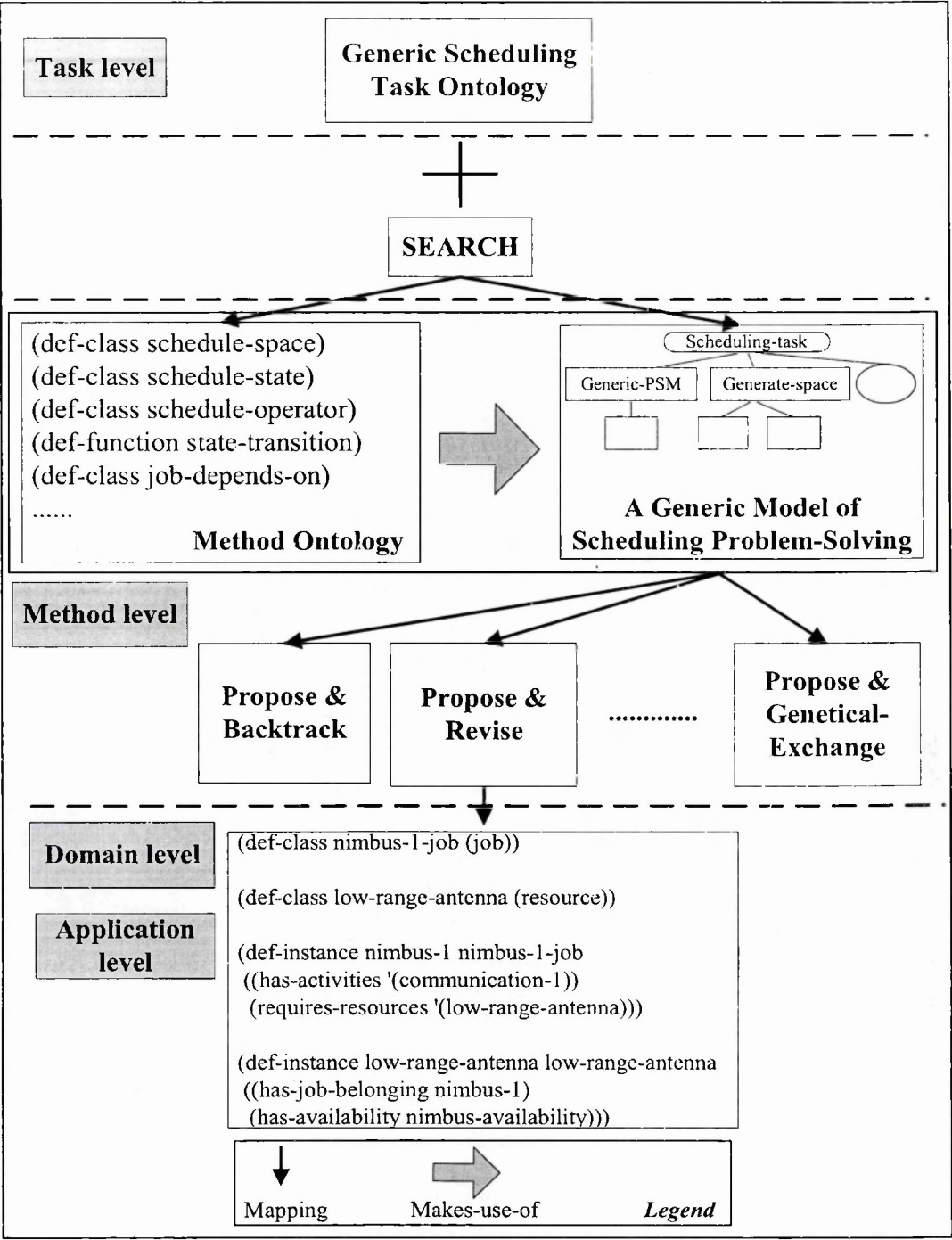


Figure 4.1. Architecture of the scheduling library by instantiating the TMDA framework.

In the following section, we describe the architecture of our library defined in terms of the TMDA framework.

4.3 Library architecture

As depicted in Figure 4.1, the construction of our library can be seen as a four-tier hierarchy, whereby we first formalise the scheduling task by defining its task ontology, and then, we define a generic model of scheduling problem-solving. More specific knowledge-intensive PSMs are defined simply by reusing and specialising the high-level tasks and methods defined in the generic model of scheduling problem-solving. Finally, to confirm

its generic nature and to evaluate its performance we apply our library to tackle scheduling applications from different domains. In the following sections, we describe each level of our library construction process.

#### 4.3.1 The task component: a generic scheduling task ontology

At this level, we develop a generic scheduling task ontology, which takes as input all the input parameters necessary to formalise the scheduling task and generates as an output a schedule. Our task ontology is generic because it does not subscribe to any particular application domain or problem-solving paradigms.

#### 4.3.2 Search as problem-solving paradigm

The space of scheduling problem-solving can be represented by means of a state-space and operators. The former indicates a problem space associated with the scheduling task. A problem space can be conceived as a constellation of states, where each state is uniquely represented by a schedule associated with it. In an initial state a schedule is incomplete because all the jobs and activities are still unassigned while in the solution state it satisfies all solution criteria. For a schedule to be a valid solution various conditions can be imposed, such as it should be complete, should not violate any constraints, should maintain all the requirements, and should be cheaper than other states. In a problem space, a transition from an initial state to a solution state can be achieved by means of operators, where each operator is responsible for assigning jobs and activities to resources and time ranges. As depicted in Figure 4.2, to construct a schedule, a search proceeds in a top-down manner: in each state transition a new job is selected, the relevant operators applicable to a job are added, and the assignment of a job is performed. Finally, a deadend state is a state from which no solution can be achieved.

A comparative discussion between the search-based and constraint-based approach to scheduling problem-solving can be found in Chapter 6.

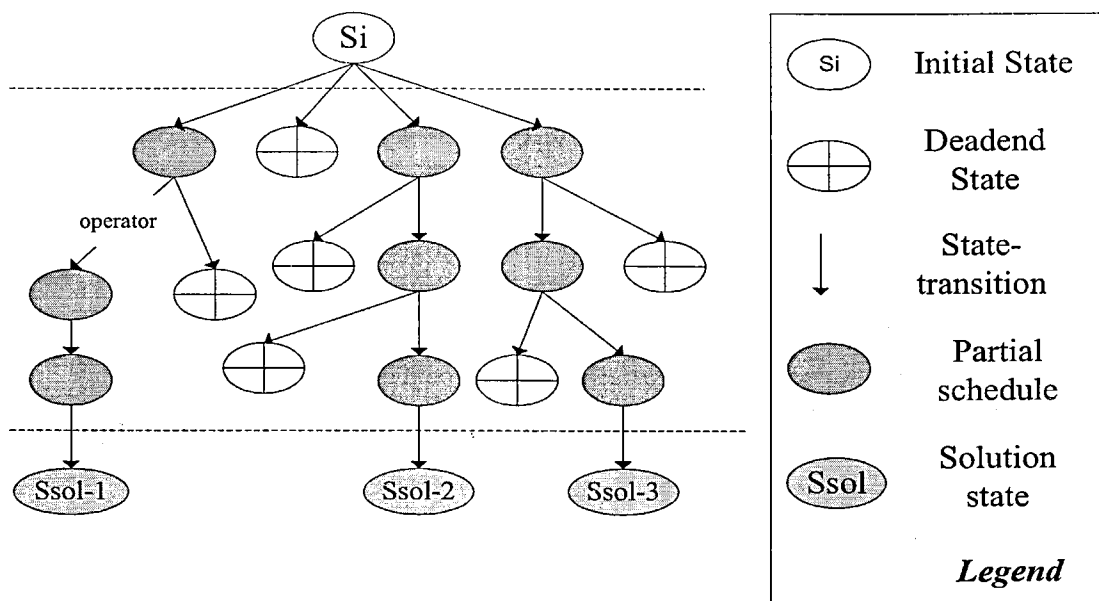


Figure 4.2. The search-based problem space of scheduling.

#### 4.3.3 The method component

The method component is the second building-block of our library and is divided into the following two components: a generic model of scheduling problem-solving and different knowledge-intensive PSMs.

First, we develop a generic model of scheduling problem-solving. This model takes as an input the appropriate concepts from the scheduling task ontology and the ‘search’ problem-solving paradigm (Newell and Simon, 1976). A generic method ontology (Musen *et al.*, 1994; Coelho and Lapalme, 1996) provides vocabulary necessary to characterise the search-based behaviour of generic schedulers. In contrast with the more specific problem-solvers of scheduling, such as Propose and Exchange (Poeck and Puppe, 1992), which imposes additional ontological commitment to model the different phases involved in their framework, a generic model imposes minimal ontological commitment by abstracting only those high-level tasks which are embedded in the specialised PSMs and are essential to construct a complete schedule. Moreover, the generic model of scheduling subscribes to the top-down approach of problem-solving, whereby the top-level scheduling task is decomposed into finite number (sub-) tasks and (sub-) methods are proposed to achieve these tasks. These tasks and methods represent the inferences that are necessary to execute the reasoning actions for constructing a schedule. Such a breakdown is not only instrumental in identifying all the generic tasks required to characterise the *scheduling* task, but also provides a generic base structure for the entire library. The schedule

construction in generic model is achieved in terms of the following two control regimes<sup>1</sup>: method independent and method specific. The former is a generic control regime and all the PSMs in our library subscribe to it, whereas the latter is a more specific control structure whose ultimate aim is to construct a complete schedule by assigning jobs to resources and time ranges.

By reusing and specialising the high-level tasks and methods defined in the generic model of scheduling problem-solving more specialised PSMs can then be constructed. This uniform engineering approach allows us to compare and contrast the knowledge requirements of these PSMs. All the PSMs in our library are constructed by specialising the control regimes and generic notions, such as *context*, *focus*, *state selection*, and *operator selection* defined in the generic model of scheduling problem-solving. Here, the notion of *context* specifies the primary function of each problem-solving phase of a PSM that needs to be carried out to construct a solution. The notion of *focus* exemplifies those variables in the problem formulation which are under scrutiny during each problem-solving phase of a PSM, and these variables must be grounded to construct a valid solution. For instance, the Propose & Exchange method (Poeck and Puppe, 1992) distinguishes between the *propose* phase and the *exchange* phase. The *context* in the former phase is to *extend* a schedule and a *focus* is on one of the unassigned jobs, whereas in the latter phase a *context* is to *revise* a schedule by fixing the constraint violations and a *focus* is on the violated constraints. A schedule extension in the *propose* phase is achieved by the *schedule-extension-operator*, which assign jobs to resources and time ranges, whereas the constraint violations are fixed by defining an *exchange-operator*. Because our library aims at providing a comprehensive framework for defining knowledge-intensive PSMs, it can cover and reason about all the validation areas crucial to scheduling, such as completion, constraint violation, requirement violation, and optimisation.

#### 4.3.4 Development of scheduling applications from different domains

To confirm the generic nature of our library we develop applications from different scheduling domains. The applications that will be used for validating our library will be chosen to cover the three major categories of scheduling: pure scheduling, resource

---

<sup>1</sup> Modelling the problem-solving behaviour involves more than making statements and describing entities in the world. Control regimes are required to specify actions and describe the order in which these are executed. OCML supports the specification of sequential, iterative, and conditional control structures by means of a number of control term constructors such as *repeat*, *loop*, *do*, *if* and *cond*, among others.

allocation, and joint scheduling (cf. Section 2.1). The validation process will involve the following three stages: 1) instantiating the task ontology with the application-specific knowledge to formalise the nature of an application; 2) selecting and configuring domain-independent PSMs from the library with respect to various domains and applications; and 3) evaluating the performance of the resulting application systems as well as an extent to which a selected PSM satisfies the needs of an application.

#### 4.4 OCML as a knowledge modelling tool

Here, we introduce the knowledge modelling language used to implement our library, i.e. the Operational Conceptual Modelling Language (OCML)<sup>2</sup> (Motta, 1999). OCML can be used to support different knowledge modelling approaches, such as CommonKADS (Wielinga *et al.*, 1992; Schreiber *et al.*, 1994) or Components of Expertise (Steels, 1990), it is primarily developed to provide a concrete modelling support for the TMDA framework. Moreover, because our library has been implemented by using OCML it provides a support for executing the definitions as well as export mechanism to other representations, including Ontolingua (Farquhar *et al.*, 1997) and OWL (McGuinness and van Harmelen, 2004).

The OCML knowledge modelling language was originally developed in the context of the VITAL project to provide an operational modelling capability for the VITAL workbench (Domingue *et al.*, 1993). OCML supports knowledge-level modelling specification of (Newell, 1982; Fensel and van Harmelen, 1994) by supporting the classes, relations, instances, functions, rules, etc. A *base ontology* provides a basic foundation for ontology development and it includes the following modules:

- **Meta:** It defines the concepts necessary to describe the OCML language, such as expressions, functional term, rule, relation, function, assertion, etc.
- **Functions:** It defines the concepts associated with function specification, such as domain, range, unary and binary-relations.
- **Relations:** It defines the concepts associated with relation specification, such as the universe and the extension of a relation, partial and total order.
- **Sets:** It defines the constructs associated with and necessary to define sets, e.g., empty set, union, intersection, exhaustive-subclass-partition, cardinality.

---

<sup>2</sup> A reference guide to OCML can be found in Appendix 4.

- **Numbers:** It defines the concepts and mathematical operations required to model mathematical calculations with numbers.
- **List:** It defines the concepts necessary to represent and manipulate lists, e.g., list, atom, first, rest, append.
- **Strings:** It defines the concepts associated with strings, e.g., string append.
- **Mapping:** It describes the concepts necessary to specify mapping mechanisms, e.g., maps-to, meta-reference, domain-reference, and so forth.
- **Frames:** It defines the concepts associated with a frame-based representation of constructs. It includes classes such as class and instance, functions like direct-instances and all-slot-values, and relations like has-one, has-at-most.
- **Inferences:** It supports all the inference mechanisms to define functions and relations.
- **Environment:** It provides an environmental support to construct OCML models and includes special operators like `exec`, which invokes a procedure from a rule and a procedure such as output to print a message.
- **Task-Method:** It provides an ontology necessary to specify tasks and PSMs.

## 4.5 Conclusion

In this chapter we have described the architecture of our library which can be realised as a four-level hierarchy by instantiating the TMDA knowledge modelling framework. At the task level, we formalise the nature of the scheduling task by constructing a generic task ontology, and then at the method level we construct a generic model of scheduling problem-solving, which takes as the input appropriate concepts from the task ontology and the search as a problem-solving paradigm. While constructing a generic model of scheduling problem-solving we develop a method ontology that provides a lexicon necessary to characterise search based problem-solving for the scheduling task. A generic model of scheduling problem-solving abstracts high-level tasks and methods that can be used to construct more specific scheduling problem-solvers. By reusing and specialising high-level tasks and methods different knowledge-intensive PSMs can be constructed. In this chapter we also introduced the knowledge modelling language that will be used to implement our library, i.e. OCML.

In the following chapter, we describe in detail the first building-block of our library: the scheduling task ontology.

# Chapter 5

## THE EPISTEMOLOGY OF THE SCHEDULING TASK

In this chapter we describe the first building block of our library: a generic task ontology specifying the space of scheduling problems.

As discussed in Chapter 2 (cf. Section 2.3.1), as a first approximation we can say that scheduling deals with the *temporally bound assignment of jobs to resources and time ranges*. This time-centric dimension distinguishes scheduling from other synthesis tasks, such as planning, design, configuration, etc. (Wielinga and Schreiber, 1997; Mittal and Frayman, 1987). A more complete definition of the scheduling task can be given as follows:

*“An assignment of time-constrained jobs to time-constrained resources within a pre-defined time framework, which represents the complete time horizon of a schedule. Normally an admissible schedule must not violate any of the constraints imposed on jobs or resources and must satisfy all the input requirements. More in general, the output of the scheduling task is a legal schedule in accordance with a given solution criterion (e.g., complete, admissible, feasible). Preference specific decisions can influence the cost of a schedule”.*

According to this definition the notions of constraint and requirement are central to scheduling (cf. Section 2.3.2). Constraints restrict the space of admissible solutions and are often of organisational or technological nature - e.g. in an airport gate scheduling (Jo *et al.*, 1997) limitations may be enforced on the priority among flights, compatibility of gates with aircrafts, area restrictions, etc. Requirements specify desired properties of a schedule. For instance, one of the requirements in the satellite-scheduling application (cf. Chapter 8) called ‘number-of-communication-slots’ states that each satellite must have at least four communication slots each day with the allocated antennas. In addition, a cost criterion may also play a vital role, as multiple solutions can be admissible for a particular problem, and some of them can be deemed to be more ‘cost-effective’ than others. For instance, in a manufacturing scenario, we may privilege solutions which maximise the throughput, or, in some other cases we may prefer solutions that minimise the ‘idle time’ of the resources.

As discussed in section 3.3, several attempts have been made in the past at developing scheduling task ontologies (Ikeda *et al.*, 1998; Smith and Becker, 1997; and Hama *et al.*, 1992a, b). These attempts have provided limited results (cf. Section 3.4.2), as in some cases they subscribe to specific scheduling domains, algorithms, or ‘scheduling shells’, or

in some other cases fail to provide the level of detail and formalisation required to characterise the scheduling task precisely. Moreover, important ontological distinctions are also missing from these proposals. In a nutshell, no comprehensive analysis exists, which provides a formal account of the scheduling problem, independently of the way scheduling problems can be approached. Thus, our main aim here is to put the scheduling task on firm ontological foundations and provide both an adequate theoretical analysis of the problem and a concrete engineering resource, which can be used to model specific scheduling problems. Our task ontology is generic because it does not subscribe to any particular application domain or problem solving approach. Finally, while developing a task ontology we also take into account characteristics that are unique to the different problem types of scheduling (cf. Section 2.1).

This chapter is organised as follows. In the following section, we provide a generic specification of the scheduling task and also formulate different criteria to validate solutions to a task. In section 5.2, we provide a more detailed specification of the ontology. In section 5.3, we compare our work with existing proposals in the field, and finally in section 5.4 we draw the main conclusions from this chapter.

## 5.1 A generic specification of the scheduling task

In our framework, the scheduling task is formally represented as a mapping from a nine-dimensional space:  $\mathbb{J}, A, R, Tr, C, Req, Pr, Cf, Cr$  to a schedule,  $S$ . These parameters are described below.

- *Jobs,  $J = \{j_1, \dots, j_m\}$*  A set of jobs to be assigned to a set of resources for their execution.
- *Activities,  $A$* . For each job,  $j_i$ , there are  $n$  uniquely consisted of activities. The set of all such activities is denoted as  $A_i = \{a_{i1}, \dots, a_{in}\}$
- *Resources,  $R = \{r_1, \dots, r_p\}$*  A set of resources to which the jobs and activities can be assigned for their execution.
- *Constraints,  $C = \{c_1, \dots, c_l\}$*  A set of constraints that must not be violated by a solution schedule.
- *Requirements,  $Req = \{req_1, \dots, req_k\}$*  A set of requirements that describe the necessary properties of a solution schedule.
- *Schedule time range,  $Tr$* . The time horizon in which the schedule takes place. It is represented in terms of a start time and an end time.

- *Preferences*,  $P = \{p_1, \dots, p_n\}$  A set of criteria for choosing among competing solution schedules. Each preference defines a partial order over the set of solution schedules.
- *Cost function*,  $Cf$ . A function, which computes the cost of a solution schedule.
- *Solution criterion*,  $Cr$ . A mapping from a schedule  $S$  to  $\{True, False\}$  which determines whether a candidate schedule is a solution. A solution criterion normally requires  $S$  to be *correct*, *complete*, *consistent*, and *feasible* - see the following section for the definitions of these properties. More restrictive solution criteria may introduce an optimality condition based on the applicable preferences and cost-function.
- *Schedule*,  $S = \{s_1, \dots, s_n\}$  A schedule is a set of quadruples of the form,  $\langle j_m, a_{mn}, r_k, jtr_{m,n,k} \rangle$  where  $j_m$  is a job,  $a_{mn}$  is an activity associated with  $j_m$ ,  $r_k$  is a resource, and  $jtr_{m,n,k}$  is the *job time range* associated with the assignment of  $j_m$  and  $a_{mn}$  to resource  $r_k$ . The job time range is represented in terms of the earliest and latest start and end times and is a sub-interval of  $Tr$ .

#### 5.1.1 Validation criteria for a solution schedule

- $S$  is *correct*, if for every job  $j_m$  and activity  $a_{mn}$ , the pair  $\langle j_m, a_{mn} \rangle$  appears no more than once in  $S$ . This criterion is also referred to as an *occurrence constraint* (Talbot, 1982).
- $S$  is *complete*, if for each job  $j_m$  and activity  $a_{mn}$  in  $A$ , there exists a quadruple  $q$  in  $S$ , such that  $q = \langle j_m, a_{mn}, r_k, jtr_{m,n,k} \rangle$
- $S$  is *consistent*, if it does not violate any applicable constraints in  $C$ .
- $S$  is *feasible*, if it satisfies all the requirements in  $Req$ .
- $S$  is *optimal* if it is a solution schedule and no other solution schedule has a lower cost than  $S$ .

In the following section we describe the important concepts in the task ontology by providing relevant definitions in OCML.

## 5.2 The scheduling task ontology

Our scheduling ontology consists of about 106 definitions, and in addition, it relies on two underlying ontologies, Base Ontology and Simple Time<sup>1</sup>. The Base Ontology provides the definitions for basic modelling concepts, such as tasks, relations, functions, roles, numbers, and sets. Initial versions of the Simple Time ontology used Allen's (1983) representation of standard time relations to define notions, such as time point, time range, duration, calendar date. We augmented it with Zou and Fikes (2000) representation of a time point

---

<sup>1</sup> The OCML version of the complete Simple Time ontology can be found in Appendix 3.

to provide different levels of granularity, such as year, month, week, day, hour, minute, second, etc. Our Simple Time ontology also takes into account the classes needed to represent calendar months, calendar days, etc. As described in Chapter 4, the scheduling task ontology is modelled by using the OCML knowledge modelling language<sup>2</sup>, which provides support for executing the definitions in the ontology as well as export mechanisms to other representations, including Ontolingua (Farquhar *et al.*, 1997) and OWL (McGuinness and Harmelen, 2004). The OCML version of the task ontology is publicly available and can be browsed by using the WebOnto (Domingue, 1998) environment at the following URL: <http://webonto.open.ac.uk>.

### 5.2.1 Scheduling task and default schedule solution

Our task modelling framework characterises a generic task in terms of input and output roles, preconditions and a goal expression (Fensel and Motta, 2001; Motta, 1999). Having already described the input and output roles for the scheduling task –see section 5.1, here we limit ourselves to specifying the precondition and the goal<sup>3</sup>. The precondition states that jobs and resources are required for a meaningful specification. If no solution criterion is provided, then a default one is applied. The goal expression simply states that the solution criterion must hold for the output schedule.

```
(def-class SCHEDULING-TASK (goal-specification-task) ?task
  ((has-precondition :value (kappa (?task)
    (exists (?x ?y)
      (and (member ?x (role-value
        ?task 'has-jobs))
        (member ?y
          (role-value
            ?task 'has-
resources))))))
    (has-goal-expression :type binary-kappa-expression
      :default-value (kappa (?task ?schedule-model)
        (default-schedule-solution
          ?schedule-model ?task)))))
```

The default solution criterion is represented as follows:

<sup>2</sup> The OCML version of the complete task ontology can be found in Appendix 1.

<sup>3</sup> A precondition specifies what must be true before executing a goal-specification-task, whereas a goal-expression specifies the goal associated with a goal-specification-task, e.g. a goal associated with the scheduling task is to construct a valid solution schedule.

```

(def-relation DEFAULT-SCHEDULE-SOLUTION (?sc ?task)
:constraint (and (schedule-model ?sc)
                  (scheduling-task ?task))
:iff-def (and (schedule-is-correct ?sc)
              (schedule-minimally-complete ?sc
              (role-value ?task has-jobs))
              (maximally-admissible-schedule ?sc
              (role-value ?task
                          has-hard-constraints))
              (schedule-is-feasible ?sc
              (role-value ?task has-requirements))))

```

More restrictive validation criteria may specify an optimality condition on a solution schedule, but we refrain from including an optimality notion. Due to the unique specification of the optimality criterion in different scheduling domains, such as maximisation of resource utilisation or minimisation of cost, we believe that it's better to specialise the optimality criterion according to the specific scheduling domains instead of providing a single optimality criterion to evaluate a solution schedule in all the domains.

### 5.2.2 Modelling the notion of a job

The class `job` represents an entity that has a list of activities and can be assigned over available resources and time ranges for its execution. The class `job` has the following attributes.

**Has-activity:** This slot epitomises the fact that every job can have a list of activities that need to be performed in order to accomplish a job. For instance, in the manufacturing environment, a drilling job could have activities such as: drilling-machine set-up, loading of a drilling job on a drilling-machine, actual drilling operation, unloading of a drilling job from a drilling-machine, etc. The attributes of activities are basically the same as those for jobs, except that activities are not further refined into sub-activities.

**Requires-resource:** Each job requires a number of resources on which it can be assigned for their execution. This representation is similar to the one used to characterise alternative resource scheduling problems (Saucer, 1997; and Fox and Sadeh, 1990) in which each job has a set of resources to which it can be assigned, instead of having a pre-determined unique resource for its execution.

**Requires-resource-type:** In some cases we do not need to specify concrete resources for a job, but we simply want to constrain the specific type of resources that are needed to carry out a job - e.g., a machine type, a vehicle type, a specific category of personnel, etc.

**Has-time-range:** It represents a time range assigned to each job within which a job must complete its execution. A job time range is represented by the earliest and the latest start and finish times. It is represented by the slot `has-time-range`, which inherits the values of the class `job-time-range` (cf. Section 5.2.5.1).

**Has-due-date:** The calendar date by which a job must be dispatched to a customer. The violation of a due-date can have direct or indirect impact on a business, such as loss of business. The due-date of a job can also be used to determine the job priority, and a job with the earliest due-date can be given priority for its assignment.

**Has-duration:** It represents the total amount of time that has elapsed between the start and end of a job.

**Has-load:** It represents the total number of resources each job requires for its successful completion. The default value is 1.

The following box shows the OCML definition of the class job.

```
(def-class JOB () ?j
  ((has-activity :type list)
   (requires-resource :type resource :min-cardinality 1)
   (requires-resource-type :type resource-type :min-cardinality 1)
   (has-time-range :type job-time-range :max-cardinality 1)
   (has-due-date :type calendar-date :max-cardinality 1)
   (has-duration :type duration :max-cardinality 1)
   (has-load :type integer :default-value 1)))
:iff-def (exists ?task (and (scheduling-task ?task)
                             (member ?j (role-value ?task has-jobs)))))
```

In the following section we discuss those relations and functions that are necessary to represent a job assignment.

### 5.2.2.1 Relations and functions required to job assignments

In our task ontology various relations and functions are defined to accomplish the assignment of jobs to resources and time ranges. These are shown in Table 5.1.

Table 5.1. The job-specific relations and functions.

Relation Name	Explanation
Assigned-to-resource (job resource schedule)	A ternary relation between a job, a resource, and a schedule. It states that a job is assigned to a particular resource in a particular schedule.
Assigned-to-resource-type (job resource-type schedule)	A ternary relation between a job, a resource-type, and a schedule. It states that a job is assigned to a particular resource type in a particular schedule.
Assigned-to-job-time-range (job job-time-range schedule)	A ternary relation between a job, a job time range, and a schedule. It states that a job is assigned to a particular time range in a particular schedule.
Assigned-job	This checks whether a job is already assigned to a resource and a time range.
Unassigned-job	The opposite of assigned-job.

Function Name	Explanation
Resources-assigned-to-job	This function retrieves all the resources assigned to a job
Resource-types-assigned-to-job	This function retrieves all the resource-types assigned to a job
Time-range-assigned-to-job	This function retrieves a time range assigned to a job
Earliest-Start-time-of-a-job	This function retrieves the earliest start time of a job
Latest-start-time-of-a-job	This function retrieves the latest start time of a job
Earliest-end-time-of-a-job	This function retrieves the earliest end time of a job
Latest-end-time-of-a-job	This function retrieves the latest end time of a job

#### 5.2.2.2 Relations to specify the temporal ordering among jobs

In our task ontology we take into account the following five cases that can be used to impose a temporal ordering among any two jobs that are contending for the same resource. These relations can be applied on any unordered pair of jobs. The following relations can be realised on the same lines with the relations depicted in Table 2.1 (cf. Chapter 2).

- **Finishes-before ( $job_1$   $job_2$ ):** This is a binary relation between any two jobs, say  $j_1$  and  $j_2$ , which is true if the latest end time of  $j_1$  precedes the earliest start time of  $j_2$ . The relation *precedes* is inherited from the Simple Time Ontology. It is a binary relation between any two time points, say  $tp_1$  and  $tp_2$ , which states that a time point,  $tp_1$  is earlier than a time point  $tp_2$ ;
- **Job1-before-job2 ( $job_1$   $job_2$ ):** This is a binary relation between any two jobs, say  $j_1$  and  $j_2$ . It states that if the sum of the durations of a job  $j_1$  and a job  $j_2$  is greater than the difference between the latest end of a job  $j_1$  and the latest start time of a job  $j_2$ , and if it is less than the difference between the latest end time of a job  $j_2$  and the earliest start time of a job  $j_1$ , then a job  $j_1$  is assigned before a job  $j_2$ . The arithmetic equation that is used in this relation is indicated as:  $(\text{latest-end-time of } j_1 - \text{earliest-start-time of } j_2) < \text{duration-of-job}_1 + \text{duration-of-job}_2 \leq (\text{latest-end-time of } j_2 - \text{earliest-start-time of } j_1)$ ;
- **Job2-before-job1 ( $job_1$   $job_2$ ):** This is a binary relation between any two jobs, say  $j_1$  and  $j_2$ . It is the inverse of the relation *job1-before-job2*;
- **No-feasible-ordering-possible ( $job_1$   $job_2$ ):** This is a binary relation between any two jobs, say  $j_1$  and  $j_2$ . It states that if the sum of durations of a job  $j_1$  and a job  $j_2$  is greater

than the difference between the latest end time of a job  $j_1$  and earliest start time of a job  $j_2$ , and it is greater than the difference between the latest end time of a job  $j_2$  and the earliest start time of a job  $j_1$ , then no feasible ordering is possible between the jobs  $j_1$  and  $j_2$ . The arithmetic equation that is used in the relation is indicated as: (duration-of-job1 +duration-of-job2) > (latest-end-time of  $j_1$  –earliest-start-time of  $j_2$ ) and (duration-of-job1 –duration-of-job2) > (latest-end-time of  $j_2$  –earliest-start-time of  $j_1$ );

- **Any-ordering-is-possible (job<sub>1</sub> job<sub>2</sub>):** This is a binary relation between any two jobs, say  $j_1$  and  $j_2$ . It states that if the sum of the durations of a job  $j_1$  and a job  $j_2$  is less than or equal to the difference between the latest end time of a job  $j_1$  and the earliest start time of a job  $j_2$ , and if it is less than or equal to the difference between the latest end time of a job  $j_2$  and the earliest start time of a job  $j_1$ , then any ordering is possible between  $j_1$  and  $j_2$ . The arithmetic equation that is used in the relation is indicated as: (duration-of-job1 –duration-of-job2) ≤ (latest-end-time of  $j_1$  –earliest-start-time of  $j_2$ ) and (duration-of-job1 +duration-of-job2) ≤ (latest-end-time of  $j_2$  –earliest-start-time of  $j_1$ ).

### 5.2.2.3 Relations to specify the job criticality

The selection of a correct job is the most important task in scheduling because it improves the efficiency of the schedule constructions process. The following relations can be used to specify a job criticality.

- **Job-precedes (job<sub>i</sub> job<sub>k</sub>):** This is a binary relation between any two jobs, say  $j_i$  and  $j_k$ , which states that a job  $j_i$  precedes a job  $j_k$ , if  $j_i$  finishes-before  $j_k$ . It is indicated by the notation ( $j_i < j_k$ );
- **Criticality-based-on-due-date (job<sub>i</sub> job<sub>k</sub>):** This is a binary relation between any two jobs, say  $j_i$  and  $j_k$ , which states that a job  $j_i$  is more critical than a job  $j_k$ , if the due-date of  $j_i$  is before the due-date of  $j_k$ ;
- **Earliest-start-time-of-a-job (job<sub>i</sub> job<sub>k</sub>):** This is a binary relation between any two jobs, say  $j_i$  and  $j_k$ , which states that a job  $j_i$  is more critical than a job  $j_k$ , if the earliest start time of  $j_i$  precedes the earliest start of  $j_k$ ;
- **Higher-priority-job (job<sub>i</sub> job<sub>k</sub>):** This is a binary relation between any two jobs, say  $j_i$  and  $j_k$ , which states that a job  $j_i$  is more critical than a job  $j_k$ , if the duration of  $j_i$  is larger than the duration of  $j_k$ ;

- **Higher-priority-job-based-on-activities ( $job_i job_k$ ):** This is a binary relation between any two jobs, say  $j_i$  and  $j_k$ , which states that a job  $j_i$  is more critical than a job  $j_k$ , if  $j_i$  has more activities than  $j_k$ .

### 5.2.3 Modelling the notion of a resource

The class `resource` represents an entity on which jobs can be assigned for their execution. The class `resource` is considered as a finite supply entity in our task ontology and it is represented by the following attributes:

**Handles-job:** It represents the specific jobs each resource can handle for its execution, e.g. `job1`;

**Handles-activity:** It represents the specific activities each resource is capable of handling;

**Has-availability:** It represents the time interval during which a resource is available to accomplish jobs. The job assignment must be performed by complying with the resource availability period. For instance, a transmitter may have to be switched off periodically for maintenance purposes.

**Has-capacity:** It represents the maximum number of jobs each resource can handle in parallel at any given time during a schedule. The aggregate capacity of a resource is represented as an *integer*.

The following box shows the OCML definition of class `resource`.

```
(def-class RESOURCE () ?r
  ((handles-job :type job :cardinality 1)
   (handles-activity :type activity :cardinality 1)
   (has-availability :type time-range :cardinality 1)
   (has-capacity :type number :default-value 1))
  :iff-def (exists ?task (and (scheduling-task ?task)
                              (member ?r (role-value ?task has-resources))))
  :constraint (or (exists ?j (and (job ?j)
                                  (handles-job ?r ?j)))
                  (exists ?a (and (activity ?a)
                                  (handles-activity ?r ?a)))))
```

The class `unary-resource` represents a resource whose maximum aggregate capacity at any given time in a schedule is at most one job. It is modelled as a subclass of the class `resource` with the additional condition that constrains the maximum capacity of a resource.

The relation `job-and-resource-time-range`, states that all the jobs assigned over a resource, say,  $r_i$ , must be completed within the availability period of  $r_i$ . Finally, the function `maximum-capacity-of-resource` retrieves all the jobs that a resource can handle at any given time in a schedule.

### 5.2.3.1 Resource-capacity axiom

In scheduling, any two jobs that share the same unary resource may generate a conflicting situation if the time ranges of these two jobs overlap. To avoid such a type of inconsistency we define an axiom named, *resource-capacity*, which states that for a given unary capacitated resource ' $r_i$ ' with capacity ' $n_i$ ' in schedule ' $s$ ', there should not exist two jobs,  $j_i$  and  $j_k$ , such that  $j_i$  and  $j_k$  require  $r_i$  and the time ranges of  $j_i$  and  $j_k$  are overlapping with each other. The following box shows the OCML definition of *resource-capacity* axiom.

```
(def-axiom RESOURCE-CAPACITY
  (forall (?ri ?sc)
    (=> (unary-resource ?ri has-capacity ?ni)
      (not (exists ?j (and (element-of (?j ?ri ?a ?jtr) ?sc)
                           (= ?all (setofall ?j2
                                                (and (element-of
                                                       (?j2 ?ri ?a2 ?jtr2) ?sc)
                                                       (job-time-ranges-overlap
                                                        (?jtr ?jtr2))
                                                       (not (= (?j ?j2))))))
                           (> (length (cons ?j ?all2)) ?ni)))))))
```

### 5.2.4 Modelling constraints and requirements

In our task ontology we distinguish between constraints and requirements, even though existing approaches (Smith and Becker, 1997; Hori *et al.*, 1995; Mizoguchi *et al.*, 1995) fail to identify such a distinction. In our approach, constraints define a property that must not be violated by a *consistent solution*, while requirements specify properties that a *feasible solution* has to satisfy. In general, not all problem constraints are necessarily applicable to a schedule, so a solution may be admissible even if some constraints are not satisfied, they simply may not be relevant. The following box shows the OCML definition of class constraint.

```
(def-class CONSTRAINT () ?c
  ((applicability-condition :default-value (kappa (?schedule-task) (true))
                           :type unary-relation))
  (has-expression :type unary-relation :cardinality 1)))

(def-class REQUIREMENT () ?req
  ((applicability-condition :default-value (kappa (?schedule-task) (true))
                           :type unary-relation))
  (has-expression :type unary-relation :cardinality 1)))
```

The slot *applicability-condition* in the definition of class constraint specifies a logical expression which has to be true for the constraint to be satisfied.

Many approaches in the literature usually distinguish between *soft* and *hard* constraints. While hard constraints must not be violated, soft constraints can be relaxed if necessary to reach a solution. In our model, constraints define *prescriptive* properties, while requirements describe *proscriptive* ones. Soft constraints in that sense are neither prescriptive nor proscriptive, but in reality what normally happens is that soft constraints are used to determine the quality of different solution schedules (Saucer, 1997; Dorn and

Slany, 1994). A solution schedule that satisfies a maximum number of soft constraints is treated as a better solution than other competing solutions. Hence, soft constraints do not concur to define the space of admissible solutions, but they instead can be used to rank solutions. For this reason we prefer to use the notion of *preference*, which is discussed in section 5.2.6.

### 5.2.5 Representing time ranges

In our task ontology time ranges are distinguished into the following two types: 1) a time range to represent the period in which a job or activity can be executed and 2) a time range to represent a schedule horizon and a resource availability period.

#### 5.2.5.1 Representing job and activity time ranges

The time range of a job or an activity represents a time window within which a job or activity has to be executed. It is represented by the following attributes:

**Has-earliest-start-time:** It represents the earliest time a particular job can start its execution;

**Has-latest-start-time:** It represents the latest time a particular job must start;

**Has-earliest-end-time:** It represents the earliest time a particular job can finish;

**Has-latest-end-time:** It represents the latest time a particular job must finish;

**Has-unit-of-time:** It simply represents the unit used to specify the time, such as second, minute, hour, etc.

#### 5.2.5.2 Representing the schedule horizon and the resource availability

The schedule horizon and a resource availability period are represented by the following attributes:

**Has-start-time:** It represents the time by which a scheduling task must start;

**Has-end-time:** It represents the time by which a scheduling task must end;

**Has-unit-of-time:** It represents the unit in which the time is specified.

### 5.2.6 Representing cost, cost function, and preference

The scheduling task not only deals with the satisfaction of constraints or maintenance of requirements, but it can also be seen as a combinatorial optimisation problem (Kempf *et al.*, 1991), where the evaluation function of a schedule, such as maximisation of throughput or minimisation of resource idle time, should be optimised. Our task ontology provides two constructs that allow us to capture the knowledge needed to rank solutions: *preferences* and *cost-function*. Preferences allow us to describe task knowledge that can be used to assess whether a solution can be regarded as better than another. For instance, in

some cases we may prefer to use one resource rather than another, even when both are suitable for a particular job. The role of preferences is primarily to do with KA. They allow us to capture important task knowledge, which is clearly of a different nature from requirements and constraints. Once the relevant preferences are acquired we use the notion of a *cost-function* to develop an optimisation criterion for a given scheduling problem. Generally speaking, this is a non-trivial effort, as preferences tend to be heterogeneous and they have different costs associated - e.g., it may be acceptable to violate any number of 'less important' preferences, but it may be unacceptable to violate even one 'critical' preference. Therefore, it is important to emphasise that a cost function may not necessarily be numeric and often some *non-Archimedean criterion* may be applied (Motta, 1999).

Our task ontology models preferences as binary relations, which define a partial order over schedules. The class *cost-function* is defined as a mapping from schedules to costs. A cost is modelled either as a real-number or as an n-dimensional vector. We have pointed out that the role of a cost-function is to define a single optimisation criterion, which is both consistent with and subsumes the various criteria expressed by the various preferences. In our task ontology these requirements are specified by two axioms: 1) *cost-subsumes-preferences*, 2) *cost-preference-consistency*. The first axiom states that the cost-function should enforce the partial order expressed by any relevant preference. The second axiom states that the cost function should not violate any preference. The above definitions make use of the association between a cost-function and a *cost-order relation*, which expresses the partial order defined by the cost function. The following box shows the OCML definition of class *preference* and the axioms.

```

(def-class PREFERENCE () ?p
  "A preference gives the order over two schedules."
  ((has-expression :cardinality 1 :type prefer-expression)))

(def-axiom COST-SUBSUMES-PREFERENCES
  (forall (?schedule-task1 ?schedule-task2)
    (= >
      (and (scheduling-task ?task has-preferences ?prs
                           has-cost-function ?cf)
            (has-cost-order-relation ?task ?rel)
            (member ?pr ?prs)
            (has-expression ?pr ?exp)
            (proves ?exp ~(prefer ?schedule-task1 ?schedule-task2)))
      (cheaper-schedule ?rel ?schedule-task1 ?schedule-task2))))

(def-axiom COST-PREFERENCE-CONSISTENCY
  (forall (?schedule-task1 ?schedule-task2)
    (= > (and (scheduling-task ?task has-preferences ?prs
                           has-cost-function ?cf)
              (has-cost-order-relation ?task ?rel)
              (cheaper-schedule ?rel ?schedule-task1 ?schedule-task2))
      (not (exists ?pr
                  (member ?pr ?prs)
                  (has-expression ?pr ?exp)
                  (proves ?exp ~(prefer
                                ?schedule-task2 ?schedule-task1))))))))

```

### 5.2.7 Representing a schedule

The class `schedule` represents the actual mapping of a job and its activities to resources within a time range. The class `schedule` is represented in terms of a set of job-assignment quadruples.

The class `job-assignment` models a quadruple of the form `<?job ?activity ?resource ?job-time-range>`. The following box shows the OCML definitions of class `schedule` and class `job-assignment`.

```

(def-class SCHEDULE (set) ?schedule-task
  :iff-def (and (= ?quadruples (setofall ?quadruple
                                          (element-of ?quadruple ?schedule-task)))
                (every ?quadruples job-assignment)))

(def-class JOB-ASSIGNMENT () ?quadruple
  :iff-def (and (== ?quadruple (?j ?r ?a ?jtr))
                (job ?j)
                (member ?a (has-activities ?j ?list))
                (resource ?r) (job-time-range ?jtr)))

```

With these definitions, we conclude our description of the task ontology. In the next section we compare our scheduling task ontology with other proposals in the literature.

## 5.3 Comparison with other approaches

In Chapter 3 (cf. Section 3.3) we reviewed the following scheduling task ontologies: job-assignment task ontology (Hori *et al.*, 1995; Hama *et al.*, 1992a, b), MULTIS task ontology (Mizoguchi *et al.*, 1995), and OØNE ontology (Smith and Becker, 1997). Here, we highlight the main differences between our task ontology and these proposals.

### 5.3.1 Comparison with the job-assignment task ontology

The job-assignment task ontology was developed in the context of the CAKE project by Hama *et al.* The primary difference between their task ontology and ours is that their task ontology mainly focuses on the job-assignment task, which is a sub-domain of scheduling, and therefore has limited applicability. In contrast with their task ontology, our aim is to characterise the scheduling task at a generic level such that it can be used to formalise all types of scheduling problems in different domains. Another major difference between these two approaches is that of the *level of detail and formalisation* in characterising the scheduling task. In their framework different concepts required to characterise the scheduling task are specified at a very coarse-grained level. For instance, let's consider their definitions of class `job` and class `resource`.

```
(define-class JOB (?job)
  :def (source ?job))

(define-class RESOURCE (?resource)
  :def (target ?resource))
```

If we compare these definitions, with the ones presented in our task ontology (cf. Sections 5.2.2 and 5.2.3) it is clear that the main building blocks in their ontology are heavily under-specified. More importantly, as shown in Table 3.4, important concepts such as activity, requirement, preference and cost are missing from their task ontology. Finally, our task ontology provides a more comprehensive set of definitions for validating a solution schedule (cf. Section 5.1.1) whereas the job-assignment ontology does not deal with requirement violations or optimisation.

### 5.3.2 Comparison with the MULTIS task ontology

The MULTIS task ontology was developed through a task analysis interview system for a general class of scheduling tasks. The MULTIS task ontology characterises the scheduling task without subscribing to any particular application domain and in this sense it is similar to our approach. However, a few differences still exist between these two task ontologies. The primary difference is that while our task ontology provides a fine-grained characterisation of the scheduling task, the MULTIS task ontology fails to provide a complete characterisation of the scheduling task. For instance, as shown in Table 3.4, some of the important concepts, such as activity, requirement and cost are missing from MULTIS. For instance, as pointed out by a number of authors (Le Pape, 1995; Smith 1994; Fox and Sadeh, 1990) *resource-capacity* (cf. Section 5.2.3.1) is a crucial concept in scheduling, and is needed to avoid job overlapping for the utilisation of a unary capacity resource. Because such an important concept is missing from the MULTIS task ontology, it is not very clear how their framework would deal with the job overlapping situation in

scheduling. Moreover, because our task ontology clearly distinguishes between constraints, requirements, and preferences, our framework provides a detailed set of validation criteria. In MULTIS a solution schedule is validated only against completion and constraint violation. Finally, it is also not very clear how the notion of ‘job criticality’ is tackled in MULTIS because no indication is given about this.

### 5.3.3 Comparison with the OZONE ontology

The OZONE ontology assumes a constraint-directed search architecture (Lessila *et al.*, 1996; Smith, 1994). OZONE also provides a model of the scheduling task, which is defined in terms of five base concepts: *demand*, *activity*, *resource*, *product* and *constraint*. In terms of our framework the concept *product* does not directly contribute to the specification of the scheduling problem, but it can be seen as an external environmental factor. In contrast with OZONE, we are mainly interested in investigating the core issues involved in the scheduling task. The concept *activity* in OZONE has attributes such as *time range* and *assigned-resource*, but they do not deal explicitly with the *load* factor indicating the number of resources that are required by each activity. The load factor is particularly crucial in scheduling as it indicates how many resources can be required by an activity for its execution. Like the other two task ontologies we have examined in this Section, the OZONE ontology does not explicitly deal with the cost issues. As shown in Table 3.4, although they make use of the preferences in conjunction with soft constraints, no indication is given about how they can affect the cost of a schedule. Moreover, the lack of a cost function means that no mechanism is provided to integrate different preferences in order to discuss their relative importance and this also makes it difficult to assess the impact of preference-specific decisions on the cost of a schedule. In addition, no notion of requirement is included either.

The OZONE framework is built to support a constraint-based scheduling ‘shell’. Therefore, most of the definitions are geared to support the constraint-based problem solving approach. In contrast with this approach we do not make any assumptions about the type of problem-solving approaches that can be used to solve the problem. The disadvantage of subscribing to a particular problem solving approach is that important conceptual distinctions are not considered, if they are not directly supported by the problem solving environment - e.g. there is no distinction in OZONE between constraints and requirements.

## 5.4 Conclusion

In this chapter we have proposed a generic scheduling task ontology, which characterises the scheduling task independently of a particular application domain or problem-solving approach. This work is situated as a first building block in our scheduling library. Our task ontology aims to put the scheduling task on firm ontological and engineering foundations. On the one hand it helps us to understand the ontological nature of an important class of KB applications. At the same time it provides us with a reusable resource that can be used to acquire relevant scheduling knowledge in different domains. As discussed throughout this chapter, our task ontology includes and formally characterises a number of important conceptual distinctions that are missing from the existing approaches to formalising the scheduling task. Because our task ontology does not subscribe to any specific problem-solving technique, it provides a sound ontological foundation that can be used by alternative problem-solvers to tackle the scheduling task. It can also be used to support task modelling independently of any target shell or computational method. Our approach to formalising the scheduling task is generic with respect to the different classes of scheduling problems, which have been identified in the literature. This is an important feature, as our main goal here is to provide a generic reference model for *all* the major classes of scheduling problems, such as pure scheduling, resource allocation, and joint scheduling (cf. Section 2.1). In Chapter 8 we will prove this claim about by showing how scheduling applications from different domains can be modelled successfully by our task ontology.

In the following chapter we will describe the second building-block of our library: a generic model of scheduling problem-solving.

# Chapter 6

## A GENERIC MODEL OF SCHEDULING PROBLEM-SOLVING

In this chapter we describe the first part of the method component of our library<sup>1</sup>: a generic model of scheduling problem-solving (henceforth *Generic-Schedule*).

*Generic-Schedule*<sup>2</sup> takes as input appropriate concepts from the scheduling task ontology and *search* (Newell and Simon, 1976) as a problem-solving technique. *Generic-Schedule* subscribes to a top-down approach of schedule construction, whereby the top-level scheduling task is decomposed into a finite number of (-sub) tasks and (-sub) methods. Our main claim here is that these tasks and methods provide a generic problem solving structure for the entire library. As it will be shown in Chapter 7 a number of knowledge-intensive PSMs can be constructed simply by reusing or specialising these tasks and methods.

This chapter is organised as follows. In the following section we describe why we have chosen ‘search’ as our main problem-solving technique, rather than constraint-satisfaction. In section 6.2, we describe a generic method ontology for scheduling, which provides a vocabulary to characterise the search-based problem solving behaviour of scheduling. In section 6.3, we discuss the key tasks and methods in *Generic-Schedule*. In section 6.4, we compare our work with other proposals in the literature. Finally, in section 6.5 we draw the main conclusions from this chapter.

### 6.1 Search-based vs. constraint-based problem-solving

Traditionally, the scheduling task is solved by using constraint satisfaction (Dorndorf *et al.*, 2000; Cesta *et al.*, 1999; Beck *et al.*, 1998; Beck and Fox, 1998; Cheng and Smith, 1995; Dorn and Slany, 1994; Fox and Sadeh, 1990; and Fox, 1983). In contrast with these approaches, our library subscribe to a search model of problem solving. In what follows we justify our selection.

One of the main drawbacks of the constraint-satisfaction problem formulation, such as the *finite constraint-satisfaction problem* (Macworth, 1977) and its other instances, like *dual constraint graphs* or *joint graphs* (Kumar, 1992; Dechter and Pearl, 1985), is that

---

<sup>1</sup> As described earlier (cf. Section 4.3.3), the method component of our library is divided into the following two components: a generic model of scheduling problem-solving and a number of knowledge-intensive PSMs.

<sup>2</sup> The complete OCML specification of *Generic-Schedule* can be found in Appendix 2.

these characterisations model the problem solving as a set of binary constraints (Bacchus *et al.*, 2002). In our viewpoint, this is a very restrictive representation because it blurs important distinctions, e.g. between constraints, requirements, and preferences.

Another shortcoming of the constraint satisfaction approach is its static formulation of the problem as a *constraint network*. This network requires a prior knowledge of all the jobs, activities, or constraints involved in a problem. But the space of real-life scheduling domains is dynamic, where new jobs arrive without prior notice and must be accommodated in the existing batch for their accomplishment. The static formulation of constraint satisfaction faces the same level of inflexibility as experienced by Operations Research approaches (cf. Section 2.2) and therefore cannot deal with the dynamic nature of real-life scheduling.

Finally, as pointed out by Kumar (1992), although constraint-satisfaction algorithms are sophisticated in nature, they do not consider domain-specific knowledge while constructing a solution. Therefore, these algorithms provide very little insights into how domain knowledge can be used to improve the efficiency of the problem-solving process and quality of a solution.

## 6.2 A generic scheduling method ontology

Here, we describe the important concepts and relations in our generic method ontology.

### 6.2.1 Schedule space, schedule state, and schedule-state transition

As described in Chapter 4 (cf. Section 4.3.2), the space of scheduling problem-solving can be represented by means of a *state-space* and *operators*. The former indicates a problem space associated with the scheduling task and is represented by the class *schedule-space*. A schedule space is composed of a set of schedule states and each schedule state associated with a schedule space is represented by the class *schedule-state*. A schedule state is uniquely represented by a schedule, say  $S_{sch}$ , associated with it. In an initial state a schedule is incomplete because all the jobs are still unassigned while in the solution state it satisfies all the solution criteria. The following box shows the OCML definition of classes' *schedule-space* and *schedule-state*.

```
(def-class SCHEDULE-SPACE () ?x
  ((associated-with-task :type scheduling-task :cardinality 1)
   (has-states :type set :cardinality 1 :default-value nil))
  :constraint (=> (member ?s (the ?set (has-states ?x ?set)))
                 (schedule-state ?s)))

(def-class SCHEDULE-STATE () ?s
  ((has-schedule-model :type schedule-model)))
```

The notion of a *state transition* is crucial while constructing a schedule, because it enables a scheduling agent to transit from an initial state to the solution state. The state

transition is achieved by applying the *schedule operators*, which assign jobs to resources and time ranges. The following box shows the OCML definition of state-transition.

```
(def-relation STATE-TRANSITION (?s1 ?schedule-op ?s2)
  :iff-def (and (schedule-state ?s1 has-schedule-model ?schedule-model1)
    (schedule-state ?s2 has-schedule-model ?schedule-model2)
    (schedule-operator ?schedule-op has-body ?fun)
    (= ?schedule-model2 (call ?fun ?schedule-model1))
    (not (= ?schedule-model1 ?schedule-model2))))
```

The functions predecessor-state and successor-state retrieve respectively the predecessor and successor schedule states of a current schedule state.

### 6.2.2 Schedule operators

Each schedule operator extends a partial schedule state by assigning jobs to resources and time ranges. The class *schedule-operator* represents the most abstract type of operator in our method ontology. The basic type of *schedule-operator*, *schedule-extension-operator* decomposes into two sub-types, *schedule-extension-resource-operator* and *schedule-extension-time-range-operator*. The former type of operator takes as input unassigned jobs and generates as output a list of assignments of jobs to resources, whereas the latter type of operator takes as input unassigned jobs and generates as output a list of assignments of jobs to time ranges in a single state transition.

Both *schedule-extension-resource-operator* and *schedule-extension-time-range-operator* are further specialised into *multiple-schedule-extension-resource-operator* and *multiple-schedule-extension-time-range-operator*. These two operators assign jobs to resources and time ranges respectively by searching through multiple schedule states. Figure 6.1 depicts the classification of the schedule operators.

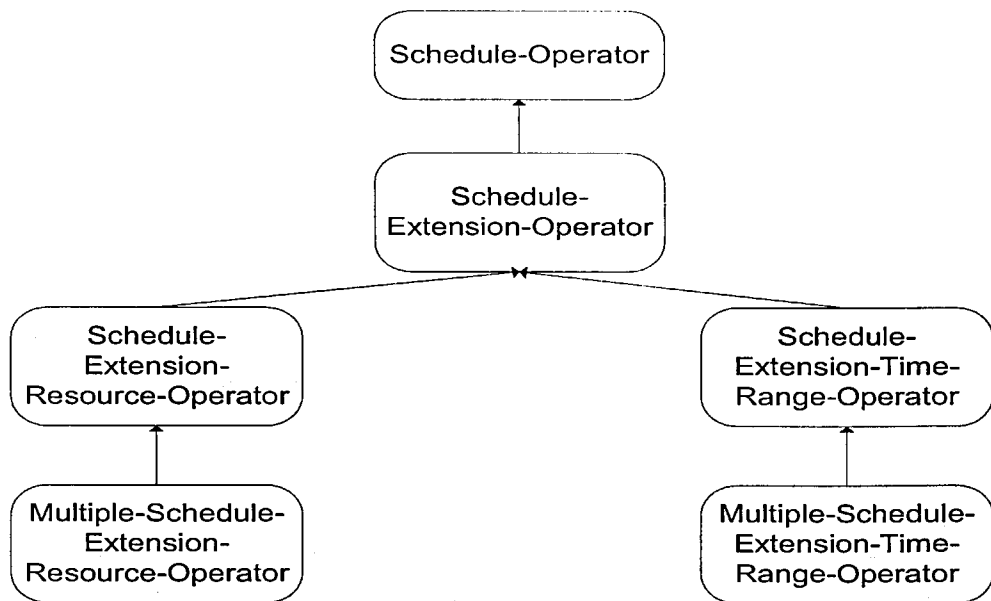


Figure 6.1 Classification of the schedule operators.

The relation `schedule-operator-order` determines the order in which different operators can be applied to accomplish a job assignment. The following box shows the OCML definition of `schedule-extension-resource-operator`.

```

(def-class SCHEDULE-EXTENSION-RESOURCE-OPERATOR (schedule-operator)
  ((applicable-to-jobs :default-value '(setofall ?x (job ?x))
    :type function-expression)
   (has-precondition :default-value (kappa (?schedule-task) (true))
    :type relation-expression)
   (has-body :type schedule-extension-resource-operator-body)))

(def-class SCHEDULE-EXTENSION-RESOURCE-OPERATOR-BODY (lambda-expression) ?x
  :no-op (:constraint (and (nth-domain ?x 1 job)
    (nth-domain ?x 2 ?sc)
    (=> (= ?z (call ?x ?j))
      (and (requires-resource ?j ?resource)
        (resource ?z))))))
  
```

### 6.2.3 Job dependency network

While constructing a schedule, a job assignment normally depends on other job assignments. To make such a job dependency explicit we construct a *job dependency network*. As pointed out by Fox (1981a) a job dependency network makes the problem-solving process of scheduling more of a ‘tightly coupled’ one, because it allows us to analyse the effects on other jobs derived from one particular job assignment. The following bullet points describe the relations and functions needed to describe a job dependency network.

- **Job-depends-on (job1, job2):** This is used to state that the assignment of a job,  $j_1$ , depend on another job,  $j_2$ ;
- **Job-affects (job1, job2):** This is the inverse of the job-depends-on relation;

- **Job-assignable (job, schedule):** This binary relation holds for a job,  $j_1$ , and a schedule,  $S$ , if  $j_1$  is an unassigned job in  $S$ , and all other jobs on which the assignment of  $j_1$  depends on are already assigned;
- **All-assignable-jobs:** This function retrieves all the unassigned jobs in a schedule;
- **Relevant-operators:** This function retrieves all the operators that can be applied to assign a particular job.

The following box shows the OCML definition of relations `job-depends-on` and `job-affects`, and function `all-assignable-jobs`.

```
(def-relation JOB-DEPENDS-ON (?j1 ?j2)
  :constraint (and (job ?j1) (job ?j2)))

(def-relation JOB-AFFECTS (?j1 ?j2)
  :constraint (and (job ?j1) (job ?j2))
  :iff-def (job-depends-on ?j1 ?j2))

(def-function ALL-ASSIGNABLE-JOBS (?js ?sc)
  :body (setofall ?x (and (member ?x ?js)
                        (unassigned-job ?x ?sc)
                        (job-assignable ?x ?sc))))
```

## 6.3 A generic problem-solving model of scheduling

As mentioned earlier, in `Generic-Schedule` the top-level scheduling task is decomposed into a number of (sub-) tasks with different (sub-) methods defined to achieve these tasks. This breakdown identifies the key knowledge-intensive tasks that are carried out when constructing a schedule. At the same time, it also provides a structure for constructing more specialised PSMs. The problem-solving process in `Generic-Schedule` uses a *method independent* control regime. This is described in the following section.

### 6.3.1 The method independent control regime

The method independent control regime, `Gen-Schedule-Control` is a high-level control loop that takes as input a list of schedule operators and a scheduling task specification, and generates as output a complete schedule. The following box shows an informal specification of `Gen-Schedule-Control`.

```

Generic-Task: Gen-Schedule-Control
Input: Schedule-Operators, Scheduling-Task
Output: Schedule-State
Control: Schedule-Space
Goal: "The output is to devise a solution-state"
Subtasks: Generate-Schedule-Space, Choose-Schedule-State, Schedule-from-State
Body: Generate-Schedule-Space (scheduling-task) -> Schedule-Space
    Repeat
    Choose-Schedule-State (schedule-space) -> Schedule-State
    IF "Choose-Schedule-State = :Nothing"
    then Return () -> :Nothing
    else
    IF "Schedule-state satisfies the goal of a scheduling task"
    then Return () -> Schedule-State
    else
    do
    Schedule-from-State (schedule-state)

```

As shown in the above box, the body of Gen-Schedule-Control first invokes the task Generate-Schedule-Space, which takes as input the scheduling task and returns either a schedule state, which satisfies the goal condition or :nothing. Having generated a schedule space, the task new-schedule-state is invoked to create a root node associated with a schedule space. A detailed discussion of the task new-schedule-state can be found in section 6.3.1.1. Once a root node associated with a schedule space is generated, then the task choose-schedule-state is invoked next, to select an appropriate schedule state for expansion. The task choose-schedule-state is discussed in section 6.3.1.2. Finally, schedule-from-state is invoked, which takes as input the schedule state selected by task choose-schedule-state and expands this schedule state by applying the relevant schedule operators. The task schedule-from-state acts as a bridge between the method independent control regime of Gen-Schedule-Control and method specific control regimes defined inside schedule-from-state. Figure 6.2 depicts the breakdown of Gen-Schedule-Control.

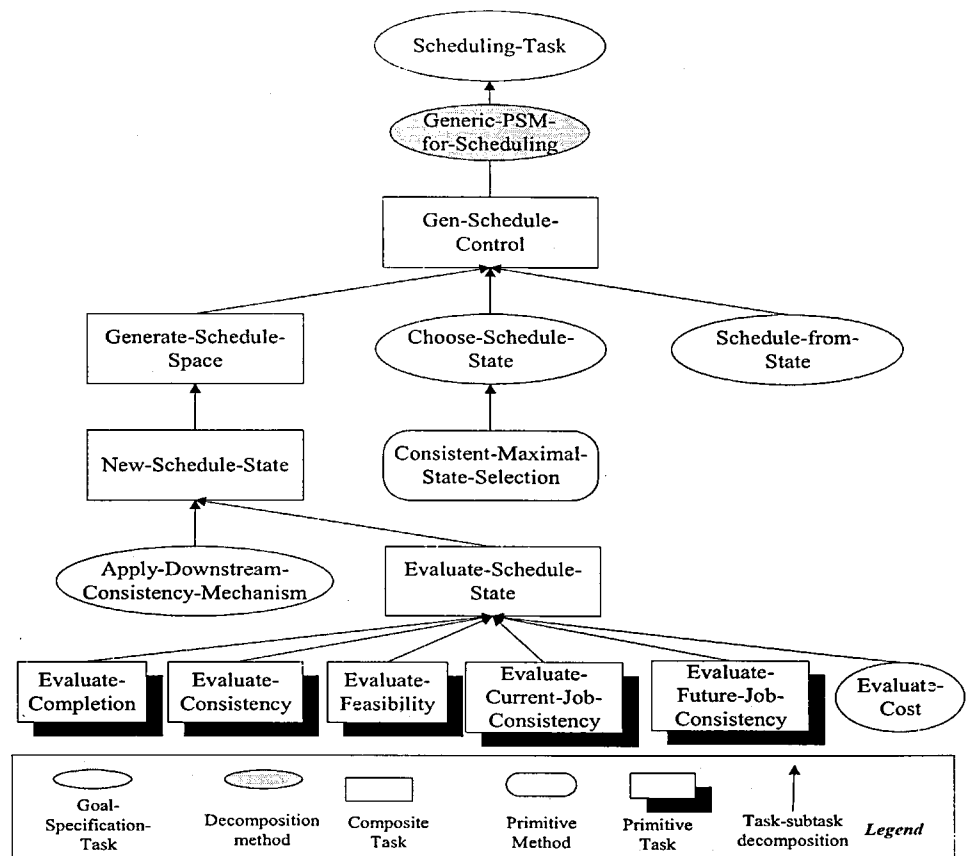


Figure 6.2. The complete breakdown of the method independent control regime.

6.3.1.1 Generation and evaluation of schedule states

Task new-schedule-state creates a root node associated with a schedule space. In each newly created schedule state we first apply the *downstream consistency enforcement* heuristic (Sadeh, 1994) by using the task apply-downstream-consistency-mechanism. This heuristic propagates the earliest start times of jobs to avoid downstream cascading constraints. The overall complexity of this heuristic is linear and in the absence of resource capacity conflict it guarantees backtrack free search.

Having applied the downstream consistency enforcement heuristic, the task evaluate-schedule-state is invoked in the body of new-schedule-state. The main purpose of this task is to evaluate each newly generated schedule state. We propose five different criteria to evaluate a schedule state and it is important to remember that these criteria are independent of each other. For instance, a PSM that does not deal with cost issues will ignore a schedule state evaluation criterion that analyses costs. The schedule state evaluation criteria are described in the following bullet points.

- **Evaluate-completion:** It checks whether a schedule associated with a state is a complete one;
- **Evaluate-hard-consistency:** It checks whether any of the constraints associated with a state are violated;

- **Evaluate-feasibility:** It checks whether all the requirements associated with a state are maintained;
- **Evaluate-cost:** It calculates the cost of a state by using the cost function (cf. Section 5.2.6);
- **Evaluate-admissibility:** Evaluating admissibility is the most difficult task in the context of a schedule state evaluation. It deals with checking whether a correct and consistent schedule state lay on a solution path. To evaluate admissibility we implemented two *look ahead* heuristics: *full looking ahead* and *partial looking ahead* (Haralick and Elliot, 1980). These two heuristics act as an oracle to anticipate the dead ends that may be encountered while constructing a schedule. The former heuristic checks the compatibility between any two unassigned jobs as well as the compatibility between the currently selected job and other assigned and unassigned jobs to ensure that the value requirements in terms of resources and time ranges of these jobs do not conflict with each other. The latter heuristic checks the value requirements compatibility between any two *unassigned* jobs.

#### 6.3.1.2 Schedule state selection

While constructing a schedule, a scheduling agent has several schedule states that can be extended to reach a solution. The main task of a scheduling agent is to select one correct schedule state from all the available schedule states such that a schedule is constructed with minimal interruptions. To this purpose, the library includes task choose-schedule-state and four different methods have been defined to achieve this task. These methods are described in Table 6.1.

Table 6.1 Different methods to select a schedule state.

Method for selecting a schedule state	Description of the method
Consistent-Maximal-Cheapest-State-Selection	This method selects a schedule state that does not violate constraints, provide maximal extension to a schedule, and has the least cost as compared to any other schedule states
Consistent-Feasible-Maximal-State-Selection	This method selects a schedule state that does not violate constraints, maintain all the requirements, and provide maximal extension to a schedule
Consistent-Cheapest-Maximal-State-Selection	This method selects a schedule state that does

	not violate any constraints, has the least cost, and provide maximal extension to a schedule
Feasible-State-Selection	This method selects a schedule state that maintain all the requirements

In Generic-Schedule, the method `consistent-feasible-maximal-state-selection` is used as a default schedule state selection strategy. It is important to keep in mind that although a schedule state is selected by using the default method, it does not affect the generic nature of Generic-Schedule. The main reason behind this is that, if a scheduling agent does not have access to additional domain knowledge, then any scheduling agent will still select a schedule state that does not violate constraints, satisfy all the requirements, and provide maximal extension to a schedule. This default method ignores all cost related issues. The following box shows the OCML definition of `choose-schedule-state` and the default method that achieves this task.

```
(def-class CHOOSE-SCHEDULE-STATE (goal-specification-task)
  ((has-input-role :value has-schedule-space)
   (has-output-role :value has-schedule-state)
   (has-schedule-space :type schedule-space)
   (has-schedule-state :type schedule-state)
   (has-goal-expression :value (kappa (?task ?s)
                                       (exists ?s
                                         (and (schedule-state ?s)
                                              (has-schedule-state ?task ?s)))))))

(def-class CONSISTENT-FEASIBLE-MAXIMAL-STATE-SELECTION (primitive-method)
  ((has-body :value (lambda (psm)
                      (in-environment
                       ((?cost-algebra . (role-value ?psm has-cost-algebra))
                        (?cost-rel . (third ?cost-algebra))
                        (?space . (role-value ?psm has-schedule-space))
                        (?states . (schedule-space-state ?space)))
                       (filter-maximal-states
                        (filter-feasible-consistent-states ?states))))))
   :own-slots ((tackles-task-type choose-schedule-state)))
```

### 6.3.2 Method specific control

The task `schedule-from-state` is a straightforward control regime, which takes as input a schedule state selected by the task `choose-schedule-state` and then expands it iteratively until a solution state is reached. The task `schedule-from-state` is achieved by the default decomposition method `expand-incomplete-state`. This is one of the most important methods in Generic-Schedule because all the PSMs in our library are constructed by specialising this method. The primary aim of the method `expand-incomplete-schedule` is to construct a complete schedule and therefore it does not deal with constraint or requirement violations or schedule optimisation issues. As a result, this method determines the *required functionality* (Fensel and Straatman, 1998) of Generic-Schedule. Also, because this method does not reason about constraint or requirement violations or schedule optimisation issues, it exhibits limited intelligence. However, as it

will be shown in Chapter 7 it is very easy to construct new knowledge-intensive PSMs, which specialises expand-incomplete-state and take into account additional types of knowledge. Figure 6.3 depicts the complete breakdown of task schedule-from-state.

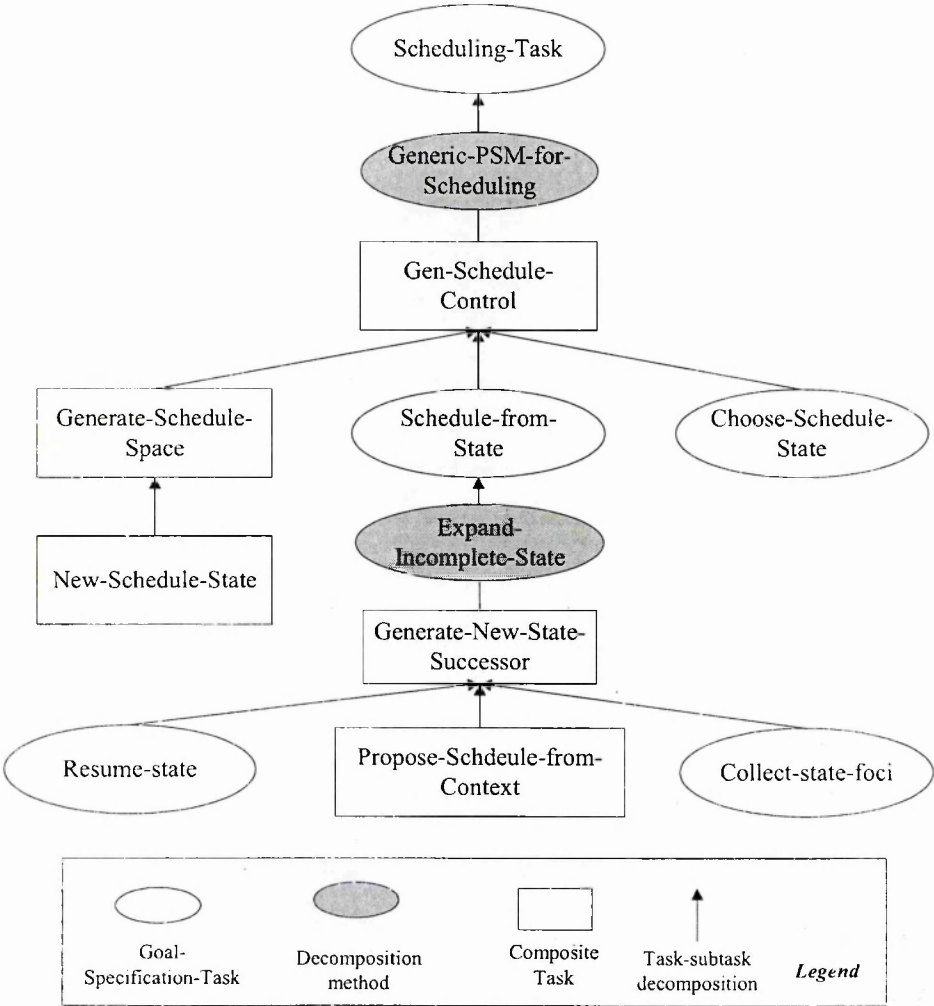


Figure 6.3. The complete breakdown of the method specific control regime.

Expand-incomplete-state takes as input the schedule state selected by choose-schedule-state and generates the successor of a current schedule state. If a successor schedule state is complete then the method returns such a schedule state as a solution state. Otherwise if the schedule state is inconsistent or infeasible then a message is issued stating that a particular schedule state is a deadend-state<sup>3</sup>. This control regime is the one that imposes minimal commitments and only uses those knowledge constructs that are defined in the scheduling task ontology.

A schedule construction in Generic-Schedule is achieved by using the notions of the *context* and *focus* (Motta, 1999). The *context* in Generic-Schedule is to **extend** a schedule and the *focus* is one of the **unassigned jobs**. However, it is important to

<sup>3</sup> The deadend-state is a problem-solving specific concept, which represents a schedule state from which a consistent solution cannot be derived.

remember that different PSMs in our library specialise the notions of context and focus. For instance, the Propose & Exchange method (Poeck and Gappa, 1993) comprises of the following two phases - the propose phase and the exchange phase. The context in the propose phase is to construct a complete schedule and the focus is one of the unassigned jobs that needs to be assigned to construct a schedule. The context in the exchange phase is to fix the constraint violations that are occurred while constructing a schedule and the focus is one of the constraint violations that need to be fixed to construct a consistent solution schedule. The following box shows an informal specification of `expand-incomplete-state` - see Appendix 2 for its OCML definition.

```

Decomposition-Method Expand-Incomplete-State
Input-Role:      Schedule-State
Output-Role:    Generates-Schedule-State
Goal:           "To extend a given input schedule state."
Subtasks:       Generate-New-Successor-State
Tackles-Task:   Schedule-from-State
Body:           If "Schedule-State violates constraints
                    tell (deadend-state ?schedule-state)
                    then Return () -> :Nothing
                    else
                      If "Schedule-State violates requirements
                        tell (deadend-state ?schedule-state)
                        then Return () -> :Nothing
                        else
                          If "Schedule-State is minimally-complete"
                            tell (solution-state ?schedule-state)
                            then Return () -> Success
                            else
                              achieve-generic-subtask
                              Generate-New-State-Successor
                                (Schedule-State
                                Schedule-Context = :Extend)

```

### 6.3.3 Generation of a successor state: generate-new-state-successor task

Task `generate-new-state-successor`, which is the main one invoked in the body of `expand-incomplete-state` and decomposes into three subtasks: `resume-state`, `collect-state-foci`, and `propose-schedule-from-context`.

Task `resume-state` is invoked in a situation where a schedule state is already extended partially and a schedule construction process needs to be resumed from this schedule state. We use a `search-control-record` to determine whether a particular schedule state has already been visited. This structure maintains dynamic problem-solving information associated with a schedule state, consisting of the schedule foci (i.e., all the unassigned jobs), currently selected schedule focus (i.e., a selected job), and all the schedule operators that can be applied to assign a focus, but still have not been used.

In a problem-solving situation, where a schedule state has not been extended yet, schedule construction is started by invoking task `collect-state-foci`. The main purpose of this task is to collect all the foci (i.e., unassigned jobs) that can be assigned to resources and time ranges. The following box shows the definition of task `collect-state-foci`.

```
(def-class COLLECT-STATE-FOCI (goal-specification-task) ?task
  ((has-input-role :value has-schedule-context
                  :value has-schedule-state)
   (has-output-role :value has-schedule-foci)
   (has-schedule-foci :type list)
   (has-schedule-context :type schedule-context)
   (has-schedule-state :type schedule-state)))

(def-class COLLECT-ASSIGNABLE-JOBS (primitive-method) ?psm
  ((has-body :value (lambda (?psm)
                      (all-assignable-jobs
                       (role-value ?psm has-jobs)
                       (the ?sc (has-schedule-model
                                (role-value ?psm has-schedule-state) ?sc))))))
   :own-slots ((tackles-task-type collect-state-foci)))
```

#### 6.3.4 Context based extension of a state: `propose-schedule-from-context`

Having collected all the foci, the task `propose-schedule-from-context` is invoked in the body of `generate-new-state-successor`. This task is a high-level control regime that takes as an input all the foci collected by the task `collect-state-foci` and then invokes the following tasks: `select-schedule-focus`, `append-search-control-record-on-focus-selection`, `collect-focus-operators`, `sort-focus-operators`, `append-search-control-record-on-focus-failure`, `generate-value-from-focus`, and `propose-schedule-from-focus` to assign the jobs from the list of foci. We will discuss these tasks through sections 6.3.5 to 6.3.8. Figure 6.4 depicts the complete breakdown of task `propose-schedule-from-context`.

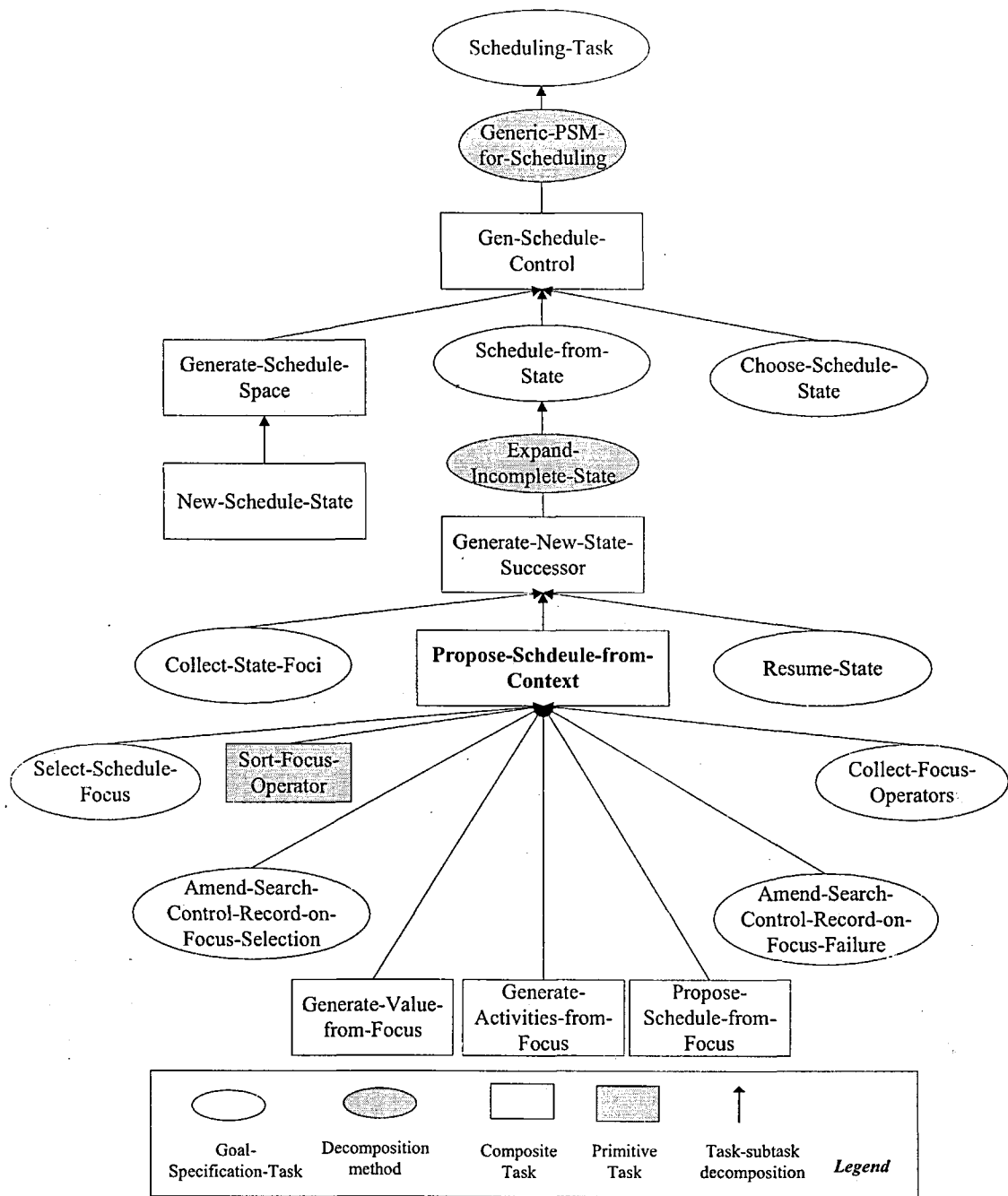


Figure 6.4. The complete breakdown of propose-schedule-from-context.

6.3.5 Correct job selection: select-schedule-focus

Selection of a correct job is the most important task while constructing a schedule, because it improves the efficiency of schedule construction by reducing unnecessary backtracking. The job selection in Generic-Schedule is achieved by the task `select-schedule-focus`, which takes as input all the foci and selects a correct focus (i.e., a candidate job). We have defined nine different methods for judiciously selecting a correct job. These methods are constructed by using job selection heuristics that were elicited both from the scheduling literature and from real-life domains. The following bullet points describe the job selection methods.

- **Job-selection-based-on-lowest-degrees-of-freedom:** This method subscribes to the *dynamic search rearrangement* heuristic (Dechter and Meiri, 1989). According to this heuristic a job that has the least number of resources and time ranges left for its assignment is selected as a candidate focus. Such a job is assumed to exhibit least *reliance*<sup>4</sup> on its resources and time ranges. If they are unavailable, then this job becomes the most likely candidate for failure;
- **Job-selection-based-on-due-date:** This method selects a job that has the earliest due date of unassigned jobs. Panwalkar and Iskander (1977) lists more than hundred job selection rules and one of the rules from their list selects a job based on its earliest due date. The fundamental difference between their rule and ours is that, in our heuristic, a job with the earliest due date is selected only when this job is competing with some other jobs for the same resource, whereas no such condition is imposed in their rule;
- **Job-selection-based-on-latest-end-time:** This method sort all the unassigned jobs based on their latest end time and then the first unassigned job from the sorted list selected as a focus;
- **Job-selection-based-on-start-time:** This method selects a job that has the earliest start time of all unassigned jobs. The method sorts all the unassigned jobs based on their earliest start times and the first job from the sorted list is selected as a focus;
- **Job-selection-based-on-precedence:** This method sorts all the unassigned jobs according to the precedence relation among them and then the first job in the sorted list is selected as a focus. We use the relation *job-precedes* (cf. Section 5.2.2.3) to impose the precedence relation among jobs;
- **Job-selection-based-on-minimal-job-dependency:** This method subscribes to the *minimal-width-ordering* heuristic (Freuder, 1982). According to this heuristic a highly constrained job is instantiated first because such a job is assumed to reduce future backtracking;
- **Job-selection-based-on-bottleneck-resources:** This method always gives priority to a job that consumes the ‘bottleneck resources’<sup>5</sup> for its accomplishment. Such a job is assumed to provide a better control in maintaining the global stability of a schedule. Because the bottleneck resources have limited capacity to execute the jobs and if the

---

<sup>4</sup> Reliance is the extent to which a particular variable must be assigned to its value such that the overall solution is formed (Beck and Fox, 1998).

<sup>5</sup> The bottleneck resources are the ones whose individual capacity determines the overall productive capacity of the scheduling process.

jobs that are using the bottleneck resources are not given priority, then it may cause such jobs to miss their due dates, which in turn requires lot of rescheduling;

- **Job-selection-based-on-number-of-activities:** This method selects a job that has the highest number of activities associated with it;
- **Job-selection-based-on-least-number-of-activities:** This method selects a job that has least number of activities associated with it.

If a scheduling application does not provide any explicit information to select a candidate focus, then the method `job-selection-based-on-lowest-degrees-of-freedom` is used as a default method to select a focus. The following box shows the OCML definition of task `select-schedule-focus` and the default job selection method.

```
(def-class SELECT-SCHEDULE-FOCUS (goal-specification-task) ?task
  ((has-input-role :value has-schedule-foci)
   (has-output-role :value has-schedule-focus)
   (has-schedule-foci :type list)
   (has-schedule-focus :type schedule-focus)
   (has-goal-expression :value (kappa (?task ?focus)
                                       (has-schedule-focus ?task ?focus)))))

(def-class JOB-SELECTION-BASED-ON-LOWEST-DEGREES-OF-FREEDOM (primitive-method)
  ((has-input-role :value has-schedule-focus-order-relation
                  :value has-possible-resources-relation)
   (has-schedule-focus-order-relation :default-value schedule-focus-order)
   (has-possible-resources-relation :default-value possible-resources-for-job)
   (has-body :value (lambda (?psm)
                      (if (= ?foci (role-value ?psm has-schedule-foci))
                          (select-most-preferred-focus
                           (collect-most-restricted-jobs
                            ?foci
                            (role-value ?psm
                                       has-possible-resources-relation))
                           (role-value
                            ?psm has-schedule-focus-order-relation))))))
  :own-slots ((tackles-task-type select-schedule-focus)))

(def-function COLLECT-MOST-RESTRICTED-JOBS (?l ?rel)
  :body (in-environment
        ((?quadruples . (sort (map '(lambda (?j)
                                      (list-of
                                       ?j (setofall ?r (holds ?rel ?j ?r))))
                                  ?l)
                              '(kappa (?x ?y)
                                      (< (length (second ?x))
                                           (length (second ?y)))))))
        (map first (filter
                  ?quadruples
                  '(kappa (?quadruple)
                          (= (first ?quadruple)
                             (first (first ?quadruples))))))))))
```

### 6.3.6 Collecting and sorting the schedule operators

Having selected the candidate focus, the tasks `collect-focus-operators` and `sort-focus-operators` are invoked in the body of `propose-schedule-from-context`. The main aim of the former task is to collect all the schedule operators that are applicable to assign resources and time ranges to the selected focus, while the latter task is used to

sort all the collected schedule operators to determine the order in which these operators are applied. The task `sort-focus-operators` make use of application-specific knowledge to prioritise the collected operators. For instance, in the satellite scheduling application (cf. Chapter 8) satellites have a fixed requirement for the antennas on which they can be assigned to perform their communication activities. All the schedule operators are sorted in such a way that only the correct antenna is assigned to establish the communication activities with a selected satellite. The following box shows the OCML definition of task `collect-focus-operators` and the method that achieves it.

```
(def-class COLLECT-FOCUS-OPERATORS (goal-specification-task) ?task
  ((has-input-role :value has-schedule-focus)
   (has-schedule-focus :type schedule-focus)))

(def-class DEFAULT-OPERATOR-COLLECTION (primitive-method) ?psm
  ((has-body :value (lambda (?psm)
                      (setofall ?op
                                (and (schedule-operator ?op
                                                            applicable-to-jobs ?l)
                                      (member (role-value ?psm 'has-schedule-focus)
                                              (eval ?l)))))))
   :own-slots ((tackles-task-type collect-focus-operators)))
```

In the following two sections we describe the resource and time range assignment of a selected focus.

### 6.3.7 Resource assignment

Once the correct focus is selected and all the operators are collected and sorted, then the task `generate-value-from-focus` is invoked in the body of `propose-schedule-from-context`. The main aim of this task is to assign resources to the selected focus. In order to accomplish this assignment, the task `generate-value-from-focus` takes as input a selected focus, collected and sorted schedule operators, and generates as output a job to which resources are assigned. The job assignment is achieved by the following two tasks: `select-resource-operator` and `try-schedule-resource-operator`. The task `select-resource-operator` takes as input all the sorted operators and selects the first operator from the sorted list. The selected operator and focus act as an input to the task `try-schedule-resource-operator`, which physically binds a selected focus to its resources to establish an assignment. The following box shows the OCML definition of task `try-resource-assignment` and function `apply-schedule-extension-resource-operator`, which is used to perform an assignment.

```

(def-class TRY-RESOURCE-ASSIGNMENT (goal-specification-task)
  ((has-input-role :value has-schedule-operator
                   :value has-schedule-focus
                   :value has-schedule-model)
   (has-output-role :value has-schedule-value)
   (has-schedule-operator :type schedule-operator)
   (has-schedule-focus :type schedule-focus)
   (has-schedule-model :type schedule-model)
   (has-schedule-value :type schedule-value)
   (has-goal-expression :value (kappa (?task ?value)
                                       (and (has-schedule-value ?task ?value)
                                             (schedule-value ?value))))))

(def-class TRY-SCHEDULE-EXTENSION-RESOURCE-OPERATOR (primitive-method)
  ((has-body :value (lambda (?psm)
                      (in-environment
                       ((?sc . (role-value ?psm 'has-schedule-model))
                        (?focus . (role-value ?psm 'has-schedule-focus))
                        (?value . (apply-schedule-extension-resource-operator
                                  ?focus ?sc
                                  (role-value ?psm 'has-schedule-operator))))
                       (if (not (= ?value :nothing))
                           (return ?value))))))
   :own-slots ((tackles-task-type try-schedule-resource-operator)))

(def-function APPLY-SCHEDULE-EXTENSION-RESOURCE-OPERATOR (?j ?sc ?op)
  :constraint (and (job ?j)
                  (schedule-model ?sc)
                  (schedule-extension-resource-operator ?op))
  :body (call (the ?body (has-body ?op ?body)) ?j ?sc))

```

### 6.3.8 Time-range assignment

Propose-schedule-from-focus is the last task that is invoked in body of the task propose-schedule-from-context. The main purpose of this task is to assign a correct time range to the selected focus. The assignment of a time range is accomplished by using the following two tasks: select-schedule-operator and try-assignment. Select-schedule-operator takes as input all the sorted operators that can be applied to assign a time range to the selected focus and the first operator from the sorted list is selected. The selected operator and focus act as an input to the task try-assignment, which generates as output a job with an assigned time range. The time range assignment is accomplished by the function called apply-schedule-extension-time-range-operator. The following box shows the OCML definition of task propose-schedule-from-focus and function apply-schedule-extension-time-range-operator.

```

(def-class PROPOSE-SCHEDULE-FROM-FOCUS (composite-task)
  ((has-input-role :value has-schedule-state
                   :value has-schedule-value
                   :value has-schedule-activity-value)
   (has-output-role :value has-output-schedule-state)
   (has-control-role :value has-schedule-model
                     :value has-schedule-operator)
   (has-schedule-state :type schedule-state)
   (has-schedule-value :type schedule-value)
   (has-schedule-activity-value :type activity-value)
   (has-output-schedule-state :type schedule-state)
   (has-body :value (lambda (?task)
                       (REPEAT
                        (in-environment
                         ((?state . (role-value ?task has-schedule-state))
                          (?record . (the-state-search-control-record ?state))
                          (?focus . (the-slot-value ?record 'has-schedule-focus))
                          (?ops . (the-slot-value ?record
                                                  'has-schedule-operators))
                          (?value . (role-value ?task has-schedule-value))
                          (?activity-value . (role-value
                                              ?task has-schedule-activity-value))
                          (?sub . (instantiate-generic-subtask
                                  ?task select-schedule-operator
                                  has-schedule-focus ?focus
                                  has-schedule-operators ?ops))
                          (?op . (solve-task ?sub))))
                        (set-slot-value ?record has-current-operator ?op)
                        (if (achieved ?sub ?op)
                            (DO
                             (set-slot-value ?record
                                              has-schedule-operators
                                              (remove ?op ?ops))
                             (in-environment
                              ((?sub2 . (instantiate-generic-subtask
                                          ?task try-schedule-operator
                                          has-schedule-operator ?op
                                          has-schedule-focus ?focus
                                          has-schedule-value ?value
                                          has-schedule-activity-value
                                          ?activity-value
                                          has-schedule-model
                                          (the-slot-value
                                           ?state 'has-schedule-model)))
                               (?result . (solve-task ?sub2)))
                              (if (achieved ?sub2 ?result)
                                  (return ?result))))
                             (return :nothing))))))
      :own-slots ((has-generic-subtasks '(select-schedule-operator
                                          try-schedule-operator))))

(def-function APPLY-SCHEDULE-EXTENSION-TIME-RANGE-OPERATOR (?j ?sc ?op)
  :constraint (and (job ?j)
                   (schedule-model ?sc)
                   (schedule-extension-time-range-operator ?op))
  :body (call (the ?body
                   (has-body ?op ?body)) ?j ?sc))

```

Once the assignment of a currently selected focus is completed then the task `try-schedule-operator` invokes the task `new-schedule-state` (cf. Section 6.3.1.1). This task repeats the complete problem-solving cycle in order to assign the remaining jobs from the list of collected foci. Once all the jobs from the collected foci are assigned then a complete schedule is returned as a solution. Finally, Figure 6.5 depicts the complete breakdown of `Generic-Schedule`.

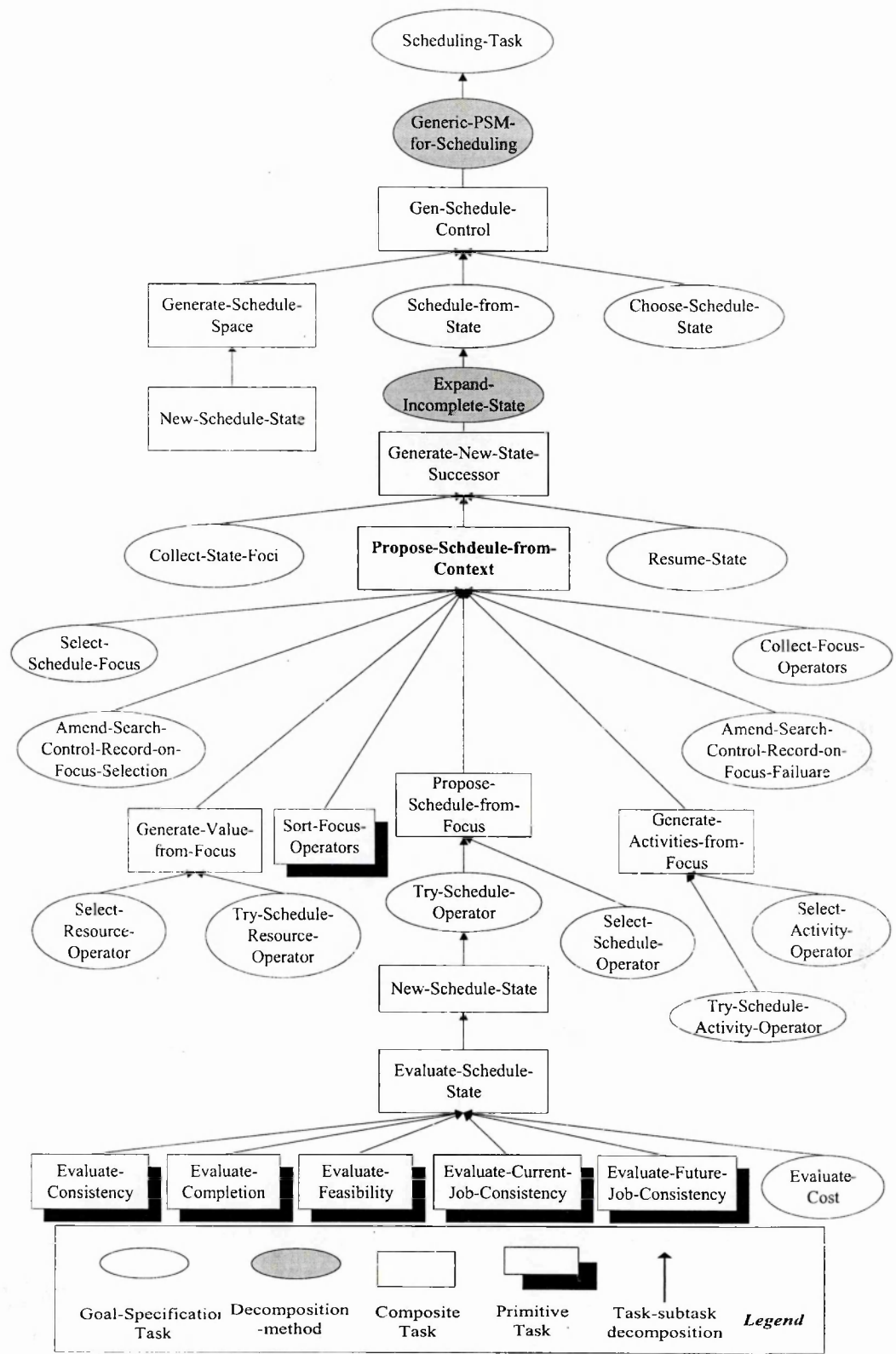


Figure 6.5. The complete breakdown of Generic-Schedule.

It is important to remember that all the tasks and methods in Generic-Schedule can be instantiated by using domain or application-specific knowledge, as with role-limiting methods (cf. Section 4.2) (Marcus, 1988). Therefore, Generic-Schedule provides a strong guidance for KA. However, compared to role-limiting methods, Generic-Schedule offers a more comprehensive and flexible framework which is not restricted by

a pre-determined sequence of questions. Thus our approach overcomes the restrictive nature of role-limiting methods (cf. Section 4.2) (Musen, 1992). Generic-Schedule now consists of 135 reusable definitions, which can be reused or specialised to construct alternative PSMs. In the next section we compare our framework with other proposals in the literature.

## 6.4 Comparison with the alternative approaches

In Chapter 3 (cf. Section 3.2) we reviewed the following scheduling libraries: production scheduling library (Hori and Yoshida, 1998), constraint-satisfaction approach (Le Pape, 1994), CommonKADS library of assignment and scheduling (Sundin, 1994), and MULTIS-II (Tijerino and Mizoguchi, 1993). Here, we highlight the main differences between our approach and these proposals.

### 6.4.1 Comparison with the domain-specific library of production scheduling

The primary difference between Hori and Yoshida's library and our approach is that we subscribe to a top-down approach of schedule construction. We start with a generic template whose components can be reused or refined to construct more specialised PSMs. In contrast with our approach, Hori and Yoshida's library subscribes to a bottom-up approach, where the knowledge requirement of all the PSMs in their library are realised entirely on the basis of processes of the production scheduling domain. This type of domain specificity restricts the reusability of their library.

In addition, in contrast with our approach, Hori and Yoshida's library fails to distinguish between method specific and method independent components. This makes it very difficult to identify how the reusability of their components can be achieved to construct new PSMs quickly. In our framework, different high-level components, such as state selection knowledge, operator construction and selection knowledge, and context and focus selection knowledge can be reused effectively for constructing new PSMs. In Chapter 7, we will prove our claim by illustrating seven PSMs constructed by specialising Generic-Schedule.

Our library follows a structured development approach and all the PSMs in our framework are constructed by subscribing to the same task and method ontology. This uniformity gives a semantic consistency to the entire library, which allows us to compare the knowledge requirements of different PSMs in the library. In contrast with our approach, different tasks in Hori and Yoshida's library use different vocabularies. For instance, *the dispatching method* (cf. Section 3.2.1) uses the notion of 'isEmpty' to check whether the list of unassigned jobs is empty while a similar kind of problem-solving action

in *the assignment method* is performed by using the notion of ‘**isDone**’. This type of semantic inconsistency makes it difficult to compare and contrast the knowledge requirements of alternative methods.

From a scheduling perspective Hori and Yoshida’s library discusses only two job selection criteria, i.e., earliest start time and down to the due-date, as compared to the broad range of job-selection criteria proposed in *Generic-Schedule* (cf. Section 6.3.5). Finally, our framework offers more exhaustive criteria to validate a solution schedule, whereby a solution schedule is validated against completion, constraint and requirement violation, and optimisation issues. In contrast with our approach a solution schedule in their library is validated only against completion and constraint violation.

#### 6.4.2 Comparison with the constraint satisfaction approach

The main difference between the ILOG library (Le Pape, 1994) and ours is that their library focuses on solving the resource allocation problem, whereas our library addresses all types of scheduling problems.

Another major difference between these two approaches is that the ILOG framework uses constraint satisfaction (CS) as the main problem-solving technique, in contrast with the knowledge-intensive approach of our library. Because of this uniform approach to modelling, CS fails to provide the fine-grained epistemological framework required to analyse the various knowledge-intensive tasks that are involved in the schedule construction process. It is essentially an implementation technique. Moreover, as discussed earlier (cf. Section 6.1) the domain-independent nature of CS techniques fails to tease out the different roles that domain knowledge plays while constructing a solution. As pointed out by Fensel and Straatman (1998), these knowledge roles provide effective means to achieve problem-solving goals and to support KA. Nevertheless, different heuristics from CS, e.g., to select a correct job (cf. Section 2.6) or to improve the search efficiency, provide important problem solving mechanisms and they have been included in our library.

#### 6.4.3 Comparison with CommonKADS

CommonKADS provides a comprehensive set of libraries, which also includes the assignment and scheduling tasks (Sundin, 1994). Analogously to the Hori and Yoshida’s library (1998), the CommonKADS library also fails to provide a clean distinction between reusable and non-reusable components. Therefore, it becomes very difficult to realise how a new PSM can be constructed simply by reusing existing tasks and methods.

More importantly, the CommonKADS library comprises only one method, i.e. Propose and Revise (Marcus and McDermott, 1989). As a result, the CommonKADS library tackles

only the completion and constraint violation issues, but cannot reason about requirement violation and optimisation issues. In contrast with the CommonKADS library, our framework provides a comprehensive coverage to tackle different schedule types.

Another limitation is that the library framework of CommonKADS is opaque, because it fails to provide the required level of detail to construct a new PSM. For instance, a job selection task in the CommonKADS library is achieved simply by sequencing all the unassigned jobs, but the knowledge sources used to sequence these jobs are not detailed. In contrast with CommonKADS, our library provides a wide range of methods for selecting and evaluating a schedule state and various job selection heuristics (cf. Section 6.3.5). Finally, our library offers a much richer framework to construct a new PSM simply by reusing the generic tasks and by specialising the notions of *context*, *focus*, *operator construction*, and *state selection* knowledge.

#### 6.4.4 Comparison with MULTIS-II

The MULTIS-II library also tackles the scheduling task at a generic level and in this sense is similar to our approach. However, some significant differences exist between these two approaches. Because a component such as *Generic-Schedule* is absent in the MULTIS-II framework, this fails to abstract high-level, reusable tasks and methods from specialised PSMs. Therefore the construction of new PSMs is very difficult in their framework. *Generic-Schedule* overcomes this problem by providing a clean separation between the method-specific and method independent components.

While our approach allows us to validate different types of schedules, a solution schedule in the MULTIS-II library is validated only against completion and constraint violation.

From a scheduling perspective, *Generic-Schedule* provides a wide range of job selection methods (cf. Section 6.3.5) to improve the efficiency of schedule construction. In contrast with our library, job selection in MULTIS-II is achieved entirely on the basis of *domain specific requirements*, which is not a very effective way to execute such an important problem-solving activity. The main reason for this is that if wrong or partial domain knowledge is used to select a job, then the job selection component may end up selecting the wrong job, which could cause heavy backtracking. In some other cases, if a scheduling domain fails to provide adequate knowledge for the job selection, then a job selection method may trivially end up selecting a first job in a queue and this job may not necessarily be the best candidate. As a result, this could deteriorate the overall quality of a schedule.

## 6.5 Conclusion

In this chapter we proposed a generic model of scheduling problem-solving called *Generic-Schedule* and a method ontology. The latter provides the vocabulary necessary to characterise the search-based problem-solving behaviour of scheduling engines. *Generic-Schedule* provides a firm theoretical and engineering foundation to scheduling problem-solving. From the theoretical perspective, *Generic-Schedule* exhibits a nice integration of the various techniques that can be used while constructing a schedule. Moreover, it also provides an insight into the different knowledge-intensive activities that take place in scheduling. From the engineering perspective, it provides a systematic abstraction of the different high-level tasks and methods, which can be reused to construct specialised PSMs.

In the next chapter we will show how different PSMs can be constructed by reusing *Generic-Schedule*.

# Chapter 7

## THE PROBLEM-SOLVING METHODS IN THE LIBRARY

Our library consists of seven PSMs: hill climbing, Propose & Backtrack (P&B) (Runkel *et al.*, 1996), Propose & Improve (P&I) (Motta, 1999), Propose & Revise (P&R) (Marcus and McDermott, 1989), Propose & Restore-feasibility (P&Rf), Propose & Exchange (P&E) (Poeck and Gappa, 1993), and Propose & Genetical-Exchange. These methods were constructed by specialising the generic model of scheduling problem solving described in Chapter 6.

The rest of the chapter is organised as follows. In the following section we describe a generic template that will be used to compare and contrast the knowledge requirements of different PSMs in our library. In section 7.2, we introduce a schedule modification operator, which deals with constraint and requirement violations and schedule optimisation issues. In section 7.3, we describe how all the PSMs in our library are constructed. Then in section 7.4, we will describe how these PSMs in our library are categorised based on the different types of schedules tackled by them. Finally, in section 7.5 we draw the main conclusions from this chapter.

### 7.1 A generic template to compare the knowledge requirements of the PSMs

In this section we describe a generic template which will be used to compare and contrast the knowledge requirements of all the PSMs in our library and which uses the generic method description framework which is discussed in Chapter 6. This generic template highlights the main types of application-specific knowledge required by a problem-solving method, say  $PSM_i$  as well as different problem-solving strategies, such as *context* and *focus* specialisation, operator and state selection, and operator configuration that are specific to each  $PSM_i$  and at the same time used by  $PSM_i$  to carry out the knowledge-intensive tasks presented while discussing *Generic-Schedule* (cf. Chapter 6).

- **Inference knowledge:** This determines what type of application-specific knowledge is required to achieve the problem-solving functionality of a PSM. For instance, a PSM that deals with requirement violations may need application-specific knowledge about how to fix them;
- **Additional subtasks:** This determines whether any new tasks or methods are required to be defined to characterise a PSM in addition to those that already exist in *Generic-Schedule*;

- **Method-specific-control-regime:** This describes how the method specific control regime of *Generic-Schedule* (cf. Section 6.3.2) is specialised in the new PSM;
- **Schedule-context:** This determines how the notion of *context* (Motta, 1999) from *Generic-Schedule* is specialised according to the different phases involved in the PSMs. For instance, the Propose & Revise method consists of the propose phase and the revise phase, and the context in the propose phase is to *extend* an incomplete schedule, whereas the context in the revise phase is to *fix* the current constraint violations;
- **Schedule-focus:** This determines how the notion of a *focus* (Motta, 1999) from *Generic-Schedule* is specialised according to the different phases involved in the PSMs. For instance, the Propose & Revise method comprises of two phases: the propose phase and the revise phase. The focus in the former phase is on the **unassigned jobs** that needs to be assigned to construct a complete schedule, whereas the focus in the latter phase is on the **constraint violations**, which need to be fixed to construct a consistent schedule;
- **Schedule focus selection strategy:** This determines how the candidate focus is selected in different PSMs;
- **Schedule operator type:** This describes which new types of operators are defined to tackle constraint and requirement violations, and optimisation issues;
- **Schedule operator order:** This determines what type of knowledge is required to rank the operators, which are applicable at any one time;
- **Schedule state selection knowledge:** This determines how the default schedule state selection policy defined in *Generic-Schedule* is specialised in different PSMs;
- **Global properties:** This field states the types of scheduling tasks that are tackled by the PSMs. For instance, whether a PSM attempts to produce optimal schedules.

Table 7.1 shows how this generic template is instantiated for *Generic-Schedule*.

Table 7.1. The knowledge requirements of *Generic-Schedule*.

Knowledge Roles	Generic-Schedule
Inference knowledge	Schedule state selection knowledge Job selection knowledge Knowledge required to determine the order in which schedule extension operators can be applied

Additional subtasks	None
Method specific control regime	Expand-incomplete-state
Schedule-context	Extend
Schedule-focus	Job
Schedule focus selection strategy	A correct focus is selected by using the application-specific knowledge; otherwise the default focus selection method called job-selection-based-on-lowest-degrees-of-freedom (cf. Section 6.3.5) from Generic-Schedule is used
Schedule operator type	Schedule-extension-resource-operator Schedule-extension-time-range-operator
Schedule operator order	Based on the focus selection
Schedule state selection knowledge	Violated constraints: No Violated requirements: No Schedule extension: Maximal
Global properties	Complete

## 7.2 The schedule modification operators

In Generic-Schedule a complete schedule is constructed by assigning jobs to resources and time ranges and it is achieved by using schedule-extension-resource-operators and schedule-extension-time-range-operators respectively. In order to deal with constraint or requirement violations we introduce a new type of operator called schedule-modification-operator. This operator is further specialised into schedule-modification-resource-operator and schedule-modification-time-range-operator. The former type of operator deals with the constraint or requirement violations occurred due to inconsistent resource assignments of jobs, whereas the latter type of operator deals with the constraint or requirement violations occurred due to conflicting time range assignments of jobs. Both types of operators can also be used to optimise a job assignment. The following box shows the OCML definition of schedule-modification-operator.

```

(def-class SCHEDULE-MODIFICATION-OPERATOR (schedule-operator)
  ((applicable-to-jobs :default-value '(setofall ?x (job ?x))
                       :type function-expression)
   (has-body :type schedule-modification-operator-body)))

(def-class SCHEDULE-MODIFICATION-OPERATOR-BODY (lambda-expression) ?x
  :no-op (:constraint (and (nth-domain ?x 1 job)
                           (nth-domain ?x 2 schedule-model)
                           (=> (= ?z (call ?x ?j ?schedule-task))
                                (or (and (schedule-model ?z)
                                         (assigned-to-resource
                                          ?j ?r ?schedule-task)
                                         (not (assigned-to-resource
                                                ?j ?r ?schedule-task))))
                                (and (schedule-model ?z)
                                     (assigned-to-job-time-range
                                      ?j ?jtr ?schedule-task)
                                     (not (assigned-to-job-time-range
                                           ?j ?jtr ?schedule-task))))))))))

```

## 7.3 Engineering of the problem-solving methods

Here, we describe how all the PSMs in our library have been constructed by reusing `Generic-Schedule`.

### 7.3.1 Hill Climbing

The hill climbing method is constructed as a straightforward refinement of `Generic-Schedule` and no additional tasks are needed. The primary difference between the hill climbing method and `Generic-Schedule` is based on the way these two methods generate a successor schedule state of a current schedule state to construct a complete schedule. While constructing a schedule, the control regime of `Generic-Schedule` generates only a single successor state of the current schedule state, whereas the hill climbing search strategy generates all the possible successors. The slot `generates-schedule-state` in the definition of the control regime of the hill climbing method states this information, which generates as an output a list of schedule states. The hill climbing method generates first a ‘good’<sup>1</sup> schedule and then it tries to optimise it. In contrast with `Generic-Schedule`, the hill climbing method performs a local exhaustive search and checks all the possible successor schedule states before selecting the next best state. The relation `locally-best-schedule-state` in the goal-expression of `hill-climbing-for-scheduling` represents the notion of locally optimal schedule state. This relation states that all the resources and time ranges that can be assigned to a job to generate an optimal assignment have already been tried, and no more optimisation is possible. As a result, the schedule state produced is the one that represents the locally optimal assignment. Finally, as in the case of `Generic-Schedule`, the hill climbing

---

<sup>1</sup> By ‘good’ schedule we mean that a solution that does not violate any constraints and maintains all the requirements.

method selects a schedule state that does not violate any constraints, maintains all the requirements, and provides maximal extension to a schedule. The following box shows the OCML definition of the method specific control regime of the hill climbing method.

```
(def-class HILL-CLIMBING-FOR-SCHEDULING (decomposition-method) ?psm
  ((has-input-role :value has-schedule-state)
   (has-output-role :value generates-schedule-states)
   (has-schedule-state :type schedule-state)
   (generates-schedule-states :type list
                              :default-value nil)
   (has-goal-expression :value (kappa (?task ?s)
                                       (locally-best-schedule-state
                                        (role-value ?task has-schedule-state))))
   (has-body :value '(lambda (?psm)
                       (in-environment
                        ((?state . (role-value ?psm has-schedule-state))
                         (?schedule-model . (the ?sc (has-schedule-model
                                                         ?state ?sc)))
                         (?constraints . (role-value
                                         ?psm has-hard-constraints))
                         (?requirements . (role-value ?psm has-requirements))
                         (?jobs . (role-value ?psm has-jobs)))
                        (if (deadend-state ?state)
                            :nothing
                            (if (constraint-violations ?state ?constraints)
                                (tell (deadend-state ?state))
                                (if (requirement-violations ?state ?requirements)
                                    (tell (deadend-state ?state))
                                    (if (state-complete ?state ?jobs)
                                        (tell (complete-state ?state))
                                        (do
                                         (achieve-generic-subtask
                                          ?psm
                                          generate-new-state-successors
                                          has-schedule-state ?state
                                          has-schedule-context :extend))))))))))
  :own-slots ((tackles-task-type schedule-from-state)
              (has-generic-subtasks generate-new-state-successors)))
```

Table 7.2 represents the knowledge requirements of the hill climbing method.

Table 7.2. The knowledge requirements of the hill climbing method.

Knowledge Roles	Hill Climbing
Inference knowledge	The hill climbing method does not require any additional inference knowledge, but exploits a cost function in more detail.
Additional subtasks	None
Method specific control regime	Hill-Climbing-for-Scheduling
Schedule-context	Extend
Schedule-focus	Job
Schedule focus selection strategy	Application-specific knowledge is used to select a focus; otherwise the default job selection method from Generic-Schedule called job-selection-based-on-lowest-degrees-of-freedom (cf. Section 6.3.5) is used

Schedule operator type	Schedule-extension-resource-operator Schedule-extension-time-range-operator
Schedule operator order	Based on the focus selection
Schedule state selection knowledge	Violated constraint: No Violated requirements: No Schedule extension: Maximal Cost: Minimal
Global properties	Complete and locally optimal

### 7.3.2 Propose and Backtrack

The Propose & Backtrack (P&B) method was proposed by Runkel *et al.* (1994) to solve the VT elevator configuration problem (Runkel *et al.*, 1996). This method is a simple refinement of Generic-Schedule and, in line with Runkel's proposal the P&B method in our library incrementally constructs a schedule by assigning jobs to resources and time ranges until an inconsistency is detected. Then it backtracks to the last consistent schedule state, where different sets of resources and time ranges are tried in order to generate a consistent assignment. This process is iterated until all the jobs are assigned without any inconsistency. The following points describe the knowledge requirements of the P&B method.

- **Inference knowledge:** This makes use of the *preference* knowledge to rank all the resources and time ranges that can be assigned to a job. This ranking mechanism can be seen as a special case of the operator preference knowledge from Generic-Schedule;
- **Additional subtasks:** No additional subtasks are defined;
- **Method-specific-control-regime:** When encountered with an inconsistent or infeasible schedule state (i.e., a schedule state violating constraints or requirements), the P&B control regime backtracks to the last consistent schedule state;
- **Schedule-context:** The schedule *context* is to **extend** a schedule until all the jobs are assigned;
- **Schedule-focus:** The schedule *focus* is one of the unassigned jobs that can be assigned to generate a complete schedule;
- **Schedule focus selection strategy:** The focus is selected by using application-specific knowledge. However, if an application fails to provide adequate knowledge to select a candidate focus, then the default focus selection method *job-selection-based-*

on-lowest-degrees-of-freedom (cf. Section 6.3.5) from Generic-Schedule is used;

- **Schedule operator type:** The jobs are assigned by using schedule-extension-resource-operator and schedule-extension-time-range-operator (cf. Section 6.2.2);
- **Schedule operator order:** The operators are ordered according to the selected focus and also by taking into account preference knowledge;
- **Schedule state selection knowledge:** This method selects a schedule state that does not violate any constraints, maintains all the requirements, and provides maximal extension to a schedule to generate a consistent solution;
- **Global properties:** The P&B method guarantees to find a complete schedule, if one exists in the problem space. It also tries to find a locally optimal solution by using preference knowledge, but because the P&B method is a greedy algorithm it is susceptible to the horizon effect.

Table 7.3 summarises the knowledge requirements of the P&B method.

Table 7.3. The knowledge requirements of the P&B method.

Knowledge Roles	Propose and Backtrack
Inference knowledge	The preference knowledge is used to rank the resources and time ranges that can be assigned to a job
Additional subtasks	None
Method specific control regime	Expand-incomplete-state
Schedule-context	Extend
Schedule-focus	Job
Schedule focus selection strategy	Focus selection is achieved by using the application-specific knowledge. Alternatively the method called job-selection-based-on-lowest-degrees-of-freedom (cf. Section 6.3.5) is used
Schedule operator type	Schedule-extension-resource-operator and schedule-extension-time-range-operator
Schedule operator order	It is determined based on the selected focus

Schedule state selection knowledge	Violated constraint: No Violated requirements: No Schedule extension: Maximal
Global properties	Complete

### 7.3.3 Propose and Improve

The main focus of both PSMs described in the previous two sections is to construct a complete schedule. However, as pointed out by Saucer (2001) and Baker (1974), in addition to the construction of a complete schedule, another important objective of the scheduling task is to optimise a complete solution over its evaluation function. To deal with optimisation issues we have included the Propose and Improve method (P&I) (Motta, 1999) in our library.

The P&I method divides a schedule construction process into the following two phases: the *propose* phase and the *improve* phase. The *context* in the former phase is to **extend** an incomplete schedule and the *focus* is on the unassigned jobs, which needs to be assigned to resources and time ranges. The context in the improve phase is to **optimise** a schedule and the focus is on the most expensive job whose assignment needs to be optimised. The propose phase of the method is constructed straightforwardly from Generic-Schedule, and therefore, in the following section we focus only on the improve phase.

#### 7.3.3.1 Modelling the P&I method

As shown in the definition of P&I in the box below, the P&I method refines generic-psm-for-scheduling in two ways - 1) a new slot called has-job-cost-function is added to represent the cost associated with the assignment of each job and 2) the goal-expression is specialised by introducing an optimality criterion. A function called job-cost-function is used to calculate the cost associated with a job assignment so that the P&I method can identify the most expensive job while optimising a complete solution. The relation P&I-Optimal in the goal-expression represents the notion of a schedule state optimality. This relation states that a schedule state is an optimal one if it is a completely expanded one, i.e. all the resources and time ranges that can be assigned to generate an optimal assignment have been tried and no more improvement is possible in the cost of the state. The relation also states that there is no other schedule state which has a lower cost than the selected schedule state. The former condition is modelled by using the relation state-fully-expanded from Generic-Schedule, whereas the latter condition is modelled by using the relation state-is-optimal. The following box shows the OCML definitions of propose-and-improve-scheduling, relation p&i-optimal, and job-cost-function.

```

(def-class PROPOSE-AND-IMPROVE-SCHEDULING (generic-psm-for-scheduling) ?psm
  ((has-input-role :value has-job-cost-function)
   (has-job-cost-function :type job-cost-function)
   (has-goal-expression :value (kappa (?psm ?state)
                                       (and (tackles-task ?psm ?task)
                                             (p&i-optimal ?state ?task)))))
  (has-output-mapping :value '(lambda (?psm ?state)
                                (the ?sc (has-schedule-model ?state ?sc))))
  (has-body :value '(lambda (?psm)
                      (in-environment
                       ((?s . (achieve-generic-subtask
                              ?psm gen-schedule-control
                              has-current-scheduling-task
                              (the ?task (tackles-task ?psm ?task)))))
                       (if (schedule-state ?s) ?s))))))
:own-slots ((has-generic-subtasks '(gen-schedule-control))))

(def-relation P&I-OPTIMAL (?state ?task)
  :iff-def (and (has-schedule-model ?state ?sc)
                (achieved ?task ?sc)
                (state-fully-expanded ?state) (state-is-optimal ?state ?task)))

(def-class JOB-COST-FUNCTION (binary-function) ?fun
  :constraint (and (nthdomain ?fun 1 ?job)
                   (nthdomain ?fun 2 ?schedule-model)
                   (range ?fun ?cost)))

```

To optimise the cost of a complete schedule the following two types of improvement operators are used: `schedule-improvement-resource-operator` and `schedule-improvement-time-range-operator`. Both the operators are defined uniformly based on `schedule-modification-resource-operator` and `schedule-modification-time-range-operator` (cf. Section 7.2).

### 7.3.3.2 The control regime of P&I

To optimise a complete solution schedule, the method specific control regime of `Generic-Schedule` is modified. The control regime of P&I is very similar to the method specific control regime of `Generic-Schedule`. The primary difference between these two control regimes is that the P&I control regime first invokes the task `generate-new-state-successor` (cf. Section 6.3.3) in the **extend** context, to construct a complete schedule. Having encountered a complete schedule state, instead of returning such a schedule state as a successor state, the task `generate-new-state-successor` is invoked again in the **improve** context to optimise a complete schedule state. The following box shows the OCML definition of the control regime of P&I.

```

(def-class PROPOSE-AND-IMPROVE-STATE (decomposition-method) ?psm
  ((has-input-role :value has-schedule-state)
   (has-output-role :value generates-schedule-state)
   (has-schedule-state :type schedule-state)
   (generates-schedule-state :type schedule-state)
   (has-body :value (lambda (?psm)
                       (in-environment
                        ((?state . (role-value ?psm has-schedule-state))
                         (?schedule-model . (the ?sc (has-schedule-model
                                                         ?state ?sc)))
                         (?constraints . (role-value ?psm has-hard-constraints))
                         (?requirements . (role-value ?psm has-requirements))
                         (?jobs . (role-value ?psm has-jobs)))
                        (if (deadend-state ?state)
                            :nothing
                            (if (constraint-violations ?state ?constraints)
                                (tell (deadend-state ?state))
                                (if (deadend-state ?state)
                                    :nothing
                                    (if (requirement-violations ?state ?requirements)
                                        (tell (deadend-state ?state))
                                        (if (state-complete ?state ?jobs)
                                            (do
                                              (tell (complete-state ?state))
                                              (achieve-generic-subtask
                                               ?psm
                                               generate-new-state-successor
                                               has-schedule-state ?state
                                               has-schedule-context :improve))
                                              (achieve-generic-subtask
                                               ?psm
                                               generate-new-state-successor
                                               has-schedule-state
                                               ?state
                                               has-schedule-context :extend))))))))))
  :own-slots ((tackles-task-type schedule-from-state)
              (has-generic-subtasks '(generate-new-state-successor))))

```

### 7.3.3.3 *Foci collection and focus selection within the improve phase*

In the improve phase, first all the assigned jobs are collected so that the cost of their assignment can be optimised. Having collected the foci, the correct focus (i.e., a job with the highest cost) is selected by using a method `select-most-expensive-job`, which achieves the task `select-schedule-focus` (cf. Section 6.3.5) from `Generic-Schedule`. In order to determine the cost associated with each job, this method makes use of `job-cost-function` (cf. Section 7.3.3.1) and the cost of a job assignment is calculated by using the function `the-most-expensive-job`. The following box shows the OCML definitions of method `select-most-expensive-job` and the function `the-most-expensive-job`.

```
(def-class SELECT-MOST-EXPENSIVE-JOB (primitive-method) ?psm
  ((has-body :value (lambda (?psm)
    (the-most-expensive-job
      (role-value ?psm has-schedule-foci)
      (the ?sc (has-schedule-model
        (role-value ?psm has-schedule-state) ?sc)
        (role-value ?psm has-job-cost-function))))))
  :own-slots ((tackles-task-type '(select-schedule-focus))
    (applicability-condition
      (kappa (?task)
        (= (role-value ?task has-schedule-context) :improve))))))

(def-function THE-MOST-EXPENSIVE-JOB (?foci ?sc ?fun) -> ?focus
  :constraint (and (list ?foci)
    (every ?foci job)
    (schedule-model ?sc)
    (job ?focus)
    (job-cost-function ?fun))
  :body (the ?focus
    (and (member ?focus ?foci)
      (= (call ?fun ?focus ?sc) ?focus-cost)
      (not (exists ?focus2
        (and (member ?focus2 ?foci)
          (= (call ?fun ?focus2 ?sc) ?focus2-cost)
          (<> ?focus2 ?focus)
          (> ?focus2-cost ?focus-cost))))))))
```

7.3.3.4 Collection and selection of the improvement operators

Once the candidate focus is selected then all schedule-improvement-operators that can be applied to optimise the cost of a selected job are collected by using a method called collect-improvable-operators. This method achieves the task collect-focus-operators (cf. Section 6.3.6) from Generic-Schedule. All the collected schedule-improvement-operators are then sorted by using the relation schedule-operator-order (cf. Section 6.2.2) and the first improvement operator from the sorted list is selected to optimise the cost of the most expensive job.

Once the cost of a currently selected focus is improved, then task new-schedule-state is invoked again. This task invokes the top-level problem-solving loop again to optimise the cost of the other jobs and if no further improvement is possible to the overall cost of a schedule then the currently optimal schedule is returned as a solution.

In total five new definitions are needed to model the P&I method. Table 7.4 summarises the knowledge requirements of the P&I method.

Table 7.4. The knowledge requirements of the P&I method.

Knowledge Roles	Propose and Improve
Inference knowledge	The schedule operator selection knowledge in both the phases The focus selection knowledge in both the phases The knowledge required to achieve the job assignment in the propose phase

	Makes a detailed use of the cost function
Additional subtasks	P&I-Optimal, job-cost-function, collect-improvable-jobs, select-most-expensive-job, collect-improvable-operators
Method specific control regime	Propose-and-Improve-State
Schedule-context	Extend, Improve
Schedule-focus	Job, most expensive job
Schedule focus selection strategy	In the propose phase, the focus is selected by using the application-specific knowledge or by using one of the job selection methods from Generic-Schedule (cf. Section 6.3.5)  The most expensive job is selected as a focus in the improve phase
Schedule operator type	Schedule-extension-operator Schedule-improvement-operator
Schedule operator order	It is determined according to a selected focus
Schedule state selection knowledge	Violated constraint: No Violated requirements: No Schedule extension: Maximal Cost: Minimum
Global properties	Complete, locally and globally optimal

### 7.3.4 Propose and Revise

The P&R method (McDermott, 1988; Marcus and McDermott, 1989) was originally developed to tackle the VT system for elevator configuration (Marcus and McDermott, 1989) and was later extended to solve the production scheduling problem (Stout *et al.*, 1988). The method was then integrated with the SALT knowledge acquisition tool (Marcus and McDermott, 1989). Several researchers (Fensel and Straatman, 1998; Wielinga *et al.*, 1995; Zdrahal and Motta, 1995; Motta, 1999) have studied the P&R method. Fensel and Straatman's (1998) work mainly aimed at analysing the competency of the P&R method. Wielinga *et al.* (1995) enumerated different assumptions and limitations of P&R in the context of the VT elevator problem. Zdrahal and Motta (1995) and Motta (1999) provide a much richer analysis of the P&R method than the two aforementioned studies and they applied the P&R method to solve parametric design problems. Their study also relates the P&R method to different constraint satisfaction techniques. Our aim here is also to provide

a uniform support for constructing the P&R method by reusing *Generic-Schedule*, while at the same time trying to tease out the characteristics that are unique to scheduling.

#### 7.3.4.1 *Initial analysis of the method*

The P&R method divides the problem-solving process into two phases: the propose phase and the revise phase. The following bullet points identify the relation between these two phases:

- The propose phase constructs a complete schedule by assigning jobs to resources and time ranges, and while constructing a schedule, it checks whether any of the constraints imposed on the schedule are violated;
- While constructing a schedule if any of the constraints imposed on the schedule are violated then the revise phase of the method is invoked to fix the constraint violations. The constraint violations are fixed by applying the least costly fix that has not been tried yet (Marcus and McDermott, 1989). However, in our approach the fixes are selected in compliance with the selected focus (i.e., constraint violation) in the revise phase;
- After the fixes are applied, the current schedule is revised tentatively to see the effects of fix application on the remaining constraint violations;
- If the constraint violations persist then the next fix from the list, which has not been tried yet, is selected;
- Finally, if no more constraints are violated then a complete and consistent schedule is established as a final solution.

In contrast with the schedule state selection policy of *Generic-Schedule* and the methods discussed in the previous three sections, the schedule state selection policy of the P&R method takes into account all those schedule states that violate constraints instead of simply ignoring such schedule states. The P&R method also specialises the notions of schedule operator and inference structure from *Generic-Schedule*:

- **Schedule operators:** The P&R method specialises the notion of a schedule operator according to the two phases involved in the method. In the propose phase *schedule-procedure* is used to assign jobs to resources and time ranges. The *schedule-procedure* is defined uniformly based on *schedule-extension-operator* (cf. Section 6.2.2). In the revise phase *schedule-fixes* are used to fix the constraint violations. The *schedule-fixes* are defined uniformly based on *schedule-modification-operator* (cf. Section 7.2). The following box shows the OCML definition of *schedule-fix*.

```
(def-class SCHEDULE-FIX (schedule-modification-operator)
  ((applicable-to-constraints :type function-expression
    :documentation "This expression returns the set of
                    constraints that can be resolved
                    by the application of this fix)))
```

- **Inference structure:** The P&R method subscribes to a knowledge-based backtracking schema (Marcus *et al.*, 1988) rather than the depth-first search with chronological backtracking search strategy of Generic-Schedule.

#### 7.3.4.2 Control regime of the P&R method

The method specific control regime of the P&R method, *propose-and-revise-control-structure*, is constructed by modifying the method specific control regime of Generic-Schedule (cf. Section 6.3.2) to deal with constraint violations. According to the original description of the method (Marcus and McDermott, 1989) the constraint violations are fixed as soon as they arise. However, in scheduling the constraints are antagonistic in nature (Stout *et al.*, 1988) mainly due to the dynamic nature of the job assignments and their inter-dependencies. As a result, in our approach the constraint violations are fixed only when a complete schedule is constructed because these violations can be dealt with simultaneously, which gives us more control to analyse the effect of fixing one constraint violation on the remaining constraint violations. *Propose-and-revise-control-structure* specialises the method specific control regime of Generic-Schedule by adding a new task *revise-schedule* to deal with the constraint violations.

*Propose-and-revise-control-structure* first invokes the task *generate-new-state-successor* in the **extend** context to construct a complete schedule. If any of the constraints are violated then they are ignored until a complete schedule is constructed. Once a complete schedule is constructed then the task *revise-schedule* is invoked in the **revise** context to fix all the ignored constraint violations. The following box shows the OCML definition of *propose-and-revise-control-structure*.

```

(def-class PROPOSE-AND-REVISE-CONTROL-STRCUTURE (decomposition-method) ?psm
  ((has-input-role :value has-schedule-state)
   (has-output-role :value generates-schedule-state)
   (has-schedule-state :type schedule-state)
   (generates-schedule-state :type schedule-state)
   (has-body :value (lambda (?psm)
                       (in-environment
                        ((?state . (role-value ?psm has-schedule-state))
                         (?sc . (the ?sc (has-schedule-model ?state ?sc)))
                         (?constraints . (role-value ?psm has-hard-constraints))
                         (?jobs . (role-value ?psm has-jobs)))
                        (if (deadend-state ?state)
                            :nothing
                            (if (requirement-violations ?state ?requirements)
                                (tell (deadend-state ?state))
                                (if (state-complete ?state ?jobs)
                                    (tell (complete-state ?state))
                                    (achieve-generic-subtask
                                     ?psm generate-new-state-successor
                                     has-schedule-state ?state
                                     has-schedule-context :extend)
                                    (if (constraint-violations ?state ?constraints)
                                        (achieve-generic-subtask
                                         ?psm revise-schedule
                                         has-schedule-state ?state))))))))))
  :own-slots ((tackles-task-type schedule-from-state)
              (has-generic-subtasks '(generate-new-state-successor
                                       revise-schedule))))

```

### 7.3.4.3 Schedule revision

The following two methods are defined in order to achieve the task `revise-schedule`: `one-step-revision-for-constraint` and `fix-constraint-monotonically`.

The method `one-step-revision-for-constraint` takes as an input an inconsistent schedule state (i.e., a schedule state violating constraints) and then invokes the task `generate-new-state-successor` in the **revise** context. This method can be used in those situations where only a single constraint is violated and therefore it has limited applicability because to fix more than one constraint violations the problem space needs to be searched in more detail so that alternative assignments can be tried for a job in conflict. Because only a single constraint is fixed by using this method no special knowledge is required to select a candidate focus. Having selected a focus, all `schedule-fixes` that can be applied to fix the selected constraint violation are collected and then sorted by the relation `schedule-operator-order` (cf. Section 6.2.2). Finally, the first `schedule-fix` from the sorted list is selected and applied to fix the constraint violation.

To deal with more than one constraint violation, the method `fix-constraint-monotonically` is included in the library. This method takes as an input a schedule state that has a number of constraint violations and then it invokes the task `generate-new-state-successor` in the **revise** context. The method `fix-constraint-monotonically` first collects all the constraint violations and then the candidate focus (i.e., constraint violation) is selected according to the relevant application-specific knowledge. Once a correct focus is selected then all the `schedule-fixes` that are applicable to fix a focus are collected and then sorted as described earlier. Having fixed a

currently selected constraint violation, this method iterates the problem-solving cycle again until all the constraint violations have been fixed. After each cycle, the task `evaluate-hard-consistency` (cf. Section 6.3.1.1) is invoked to check whether any new constraints are violated while fixing the existing ones. In both methods, application-specific knowledge is used to determine how the constraint violation can be fixed. The following box shows the OCML definitions of method `revise-schedule` and method `one-step-revision-for-constraint`.

```
(def-class REVISE-SCHEDULE (goal-specification-task) ?tsk
  ((has-input-role :value has-schedule-state)
   (has-output-role :value has-output-state)
   (has-schedule-state :type schedule-state)
   (has-output-state :type schedule-state)
   (has-goal-expression :value (kappa (?task ?s)
                                       (and (schedule-state ?s)
                                             (not (constraint-violations
                                                    ?s ?any)))))))

(def-class ONE-STEP-REVISION-FOR-CONSTRAINT (primitive-method) ?psm
  ((has-body :value (lambda (?psm)
                      (repeat
                       (in-environment
                        ((?input . (role-value ?psm has-schedule-state))
                         (?output . (achieve-generic-subtask
                                     ?psm generate-new-state-successor
                                     has-schedule-state ?input
                                     has-schedule-context :revise)))
                        (if (achieved ?psm ?output)
                            (return ?output)))))))
  :own-slots ((has-generic-subtasks generate-new-state-successor)
              (tackles-task-type revise-schedule)
              (APPLICABILITY-CONDITION
               (kappa (?task)
                       (in-environment
                        ((?input . (role-value ?task 'has-schedule-state)))
                        (= (cardinality
                           (the ?constraints (constraint-violations
                                              ?input ?constraints))) 1))))))
```

#### 7.3.4.4 Foci collection and a focus selection

All the constraint violations are collected as the candidate foci by using a method called `collect-all-constraint-violations`. This method achieves the task `collect-state-foci` (cf. Section 6.3.3) from `Generic-Schedule`.

A candidate focus (i.e., a constraint violation) is selected by using a method called `select-candidate-constraint-violation`. This method achieves the task `select-schedule-focus` (cf. Section 6.3.5) from `Generic-Schedule`. The candidate focus is selected by using application-specific knowledge, but if an application fails to provide such knowledge, then the first constraint violation from the list of collected foci is selected as a focus. The following box shows the OCML definition of method `collect-all-constraint-violations`.

```
(def-class COLLECT-ALL-CONSTRAINT-VIOLATIONS (primitive-method) ?psm
  ((has-body :value (lambda (?psm)
    (setofall ?cv
      (and (fixable-constraint ?cv)
        (member ?cv
          (the ?cs (constraint-violations
            (role-value ?psm
              has-schedule-state)
            ?cs))))))))
  :own-slots ((tackles-task-type `(collect-state-foci))
    (applicability-condition
      (kappa (?task)
        (= (role-value ?task has-schedule-context) :revise)))))
```

7.3.4.5 Collecting and selecting the fixes

Once a correct focus is selected then all the fixes that are applicable to fix the selected focus are collected by using a method called `collection-of-applicable-fixes`. This method achieves the task `collect-focus-operators` (cf. Section 6.3.6) from `Generic-Schedule`. Finally, all the collected fixes are sorted and the first fix from the list of sorted fixes is selected. The following box shows the OCML definition of method `collection-of-applicable-fixes`.

```
(def-class COLLECTION-OF-APPLICABLE-FIXES (primitive-method) ?psm
  ((has-body :value (lambda (?psm)
    (setofall ?op (and (schedule-fix
      ?op applicable-to-constraints ?l)
      (member (role-value ?psm
        'has-schedule-focus)
        (eval ?l))))))
  :own-slots ((tackles-task-type `(collect-focus-operators))
    (applicability-condition
      (kappa (?task)
        (and (= (role-value ?task has-schedule-context) :revise)
          (fixable-constraint
            (role-value ?task has-schedule-focus)))))))
```

In order to model the P&R method only six new definitions have been defined by specialising `Generic-Schedule`. Table 7.5 summarises the knowledge requirements of the P&R method.

Table 7.5. The knowledge requirements of the P&R method.

Knowledge Roles	Propose and Revise
Inference knowledge	The knowledge required to select schedule-procedure and schedule-fix The focus selection knowledge in both the phases The knowledge required to select resources and time ranges for the job assignment The knowledge required to fix the constraint violations

Additional subtasks	Revise-schedule, One-step-revision-for-constraint, Fix-constraints-monotonically, Collect-all-constraint-violations, Select-candidate-constraint-violation, Collection-of-applicable-fixes
Method specific control regime	Propose-and-revise-control-structure
Schedule-context	Extend, revise
Schedule-focus	Job, constraint violation
Schedule focus selection strategy	One of the job selection methods from Generic-Schedule (cf. Section 6.3.5) is used to select a candidate job in the propose phase Application-specific knowledge is used to select a candidate constraint violation
Schedule operator type	Schedule-procedure Schedule-fix
Schedule operator order	It is determined according to a selected focus in both the phases
Schedule state selection knowledge	Violated constraint: Minimal Schedule extension: Maximal Cost: Minimum
Global properties	Complete and consistent

### 7.3.5 Propose and Restore-feasibility

The Propose and Restore-feasibility method (P&Rf) is included in our library to deal with the requirement violations that occur while constructing a schedule. This method is similar in spirit to the Propose & Revise method, and therefore, in the following section we quickly describe how the P&Rf method is modelled by providing the pointers to the appropriate definitions in P&R.

#### 7.3.5.1 Modelling the P&Rf method

- The P&Rf method divides a schedule construction process into the *propose* phase and the *restore-feasibility* phase. The *propose* phase constructs a complete schedule by assigning jobs to resources and time ranges and if any of the requirements imposed on

a schedule are violated then the *restore-feasibility* phase is invoked to fix these violations;

- **Context, focus, and operators:** The context in the propose phase is to *extend* a schedule and the focus is on the unassigned jobs, whereas the context in the restore-feasibility phase is to *fix* all the requirement violations and the focus is on the requirement violations. In the propose phase the jobs are assigned by using *schedule-extension-operator* (cf. Section 6.2.2) and in the restore-feasibility phase a *feasibility-restoration-operator* is used to fix the violated requirements;
- **Method specific control regime of P&Rf:** The method specific control regime of P&Rf, called *propose-and-restore-feasibility-state*, can be realised along the same lines as *propose-and-revise-control-structure* (cf. Section 7.3.4.2) of P&R. The main difference between these two control regimes is that in contrast with *propose-and-revise-control-structure*, which checks for the constraint violations by using the relation *constraint-violations*, *propose-and-restore-feasibility-state* checks for requirement violations by using the relation *requirement-violations*. When encountered with a schedule state that violates requirements the task *restore-feasibility* is invoked to fix the requirement violations. This task can be realised along the same lines as *revise-schedule* (cf. Section 7.3.4.3);
- **Foci collection and a focus selection:** All the requirements that are violated while constructing a schedule are collected as the candidate foci by using the method *collect-the-requirement-violations*. This method is isomorphic to *collect-all-constraint-violations* (cf. Section 7.3.4.4). Having collected all the requirement violations, the first requirement violation from the list of collected foci is selected as a candidate focus by using a method called *select-candidate-requirement-violation*. This method can be realised on the same lines as *select-candidate-constraint-violation* (cf. Section 7.3.4.4);
- **Operator collection and selection:** All the *feasibility-restoration-operators* that can be applied to fix the selected requirement violation are collected by using a method *collection-of-feasibility-restoration-operator*. This method is similar to *collect-focus-operators* (cf. Section 7.3.4.5), and the first operator from the list of collected operators is selected.

Table 7.6 summarises the knowledge requirements of the P&Rf method.

Table 7.6. The knowledge requirements of the P&amp;Rf method.

Knowledge Roles	Propose and Restore-feasibility
Inference knowledge	<p>The knowledge required to select schedule-extension-operator and feasibility-restoration-operator</p> <p>The focus selection knowledge in both the phases</p> <p>The schedule state selection knowledge</p> <p>The knowledge required to select resources and time ranges that can be assigned to jobs in the propose phase</p> <p>The knowledge required to fix the requirement violations</p>
Additional subtasks	<p>Restore-feasibility, Focus-based-feasibility-restoration, Collect-the-requirement-violations, Select-candidate-requirement-violation, Collection-of-feasibility-restoration-operator</p>
Method specific control regime	Propose-and-restore-feasibility-state
Schedule-context	Extend, feasibility-restoration
Schedule-focus	Job, requirement violation
Schedule focus selection strategy	<p>One of the job selection methods from Generic-Schedule (cf. Section 6.3.5) are used in the propose phase to select a job</p> <p>In the restore-feasibility phase the application-specific knowledge is used to select a candidate requirement violation</p>
Schedule operator type	<p>Schedule-extension-operator</p> <p>Feasibility-restoration-operator</p>
Schedule operator order	Application-specific knowledge
Schedule state selection knowledge	<p>Requirement violations: Minimal</p> <p>Schedule extension: Maximal</p> <p>Cost: Minimum</p>
Global properties	Complete and feasible

### 7.3.6 Propose and Exchange

The Propose and Exchange (P&E) method was developed by Poeck and Gappa (1993) to tackle the assignment problem with a corresponding shell called COKE (Poeck and Puppe, 1992). The assignment problem is characterised by two types of objects: *the demand object* and *the supply object* (Baker, 1974; Sharma, 1998). The main aim of the assignment task is to map each member from the demand object set (i.e., a job) to the supply object set (i.e., a resource). Scheduling can be seen as a more complex case of the assignment task, which not only deals with the assignment of jobs to resources, but also determines the time window within which each assignment needs to take place. Therefore, we modified the original description of the P&E method to tackle the time element in scheduling.

The basic idea of the P&E method is to make locally consistent assignments until any of the constraints imposed on a schedule are violated. Once constraint violations are detected, the assignments of the conflicting jobs involved in conflict are exchanged to construct a consistent solution. Although, both the P&R and P&E methods deal with the constraint violations, the main difference between these two methods can be characterised based on how these two methods fix the constraint violations. When encountered with a constraint violation, the revise phase of P&R proposes new assignments for the jobs in conflict, whereas the exchange phase of P&E simply exchanges the assignment of the jobs involved in the constraint violations at the same depth of a search tree. If the constraint violations cannot be fixed locally then the relatively best constellation of assignments is established, and then more effort is invested to fix the remaining constraint violations.

#### 7.3.6.1 Initial analysis of the method

The P&E method divides the schedule construction process in the following two phases: *the propose phase* and *the exchange phase*. The following bullet points describe how these two phases are related with each other.

- The propose phase of the method constructs a complete schedule by assigning jobs to resources and time ranges by applying schedule-extension-operators (cf. Section 6.2.2). The context in this phase is to **extend** a schedule and the focus is on the unassigned jobs;
- If any of the constraints imposed on a schedule are violated while constructing a schedule, then the **exchange** phase of the method is invoked to fix these violations locally by exchanging the conflicting job assignments by applying exchange-operators. An exchange-operator is defined uniformly based on the definition

of schedule-modification-operator (cf. Section 7.2). The context in this phase is to **exchange** a schedule and the focus is on the constraint violations.

- If the constraint violations persist then first a complete schedule is constructed and then more effort is invested to fix the remaining constraint violations by performing global exchanges among the job assignments.

The following box shows the informal schema of the P&E method.

Input: Jobs, Resources, Time ranges  
 Output: To devise a complete and a consistent schedule

Until all the jobs are assigned, REPEAT through steps 1-3.

1. Select a job with lowest degrees of freedom;
2. Proposes a valid assignment for a selected job;
3. If the constraints are violated, then try exchanging the assignments of jobs that are in conflict to remove or minimise constraint violations;
4. If constraints are still violated, then try exchanges from a global point of view and with more effort;
5. Show the final assignment and remaining constraint violations if any.

### 7.3.6.2 The method specific control regime of P&E

Propose&Exchange-state represents the method specific control regime of P&E, which is constructed by specialising the method specific control regime of Generic-Schedule (cf. Section 6.3.2).

The propose phase begins the schedule construction process by invoking the task generate-new-state-successor in the **extend** context. It constructs a complete schedule by assigning jobs to resources and time ranges. If any of the constraints are violated while constructing a schedule then the task local-exchange-of-schedule is invoked in the **exchange** context to fix these constraint violations. Because the constraint violations are fixed as soon as they occur while constructing a solution, this strategy is similar in spirit to *extend-model-then-revise* (Motta, 1999). If not all the constraint violations can be fixed locally, then they are ignored until a complete schedule is devised. Once a complete schedule is constructed then task exchange-schedule is invoked, which tries to fix all the outstanding constraint violations by globally exchanging the job assignments involved in conflict. This type of constraint violation removal strategy is similar in spirit to *complete-model-then-revise* (Motta, 1999). The following box shows the OCML definition of propose&exchange-state.

```

(def-class PROPOSE&EXCHANGE-STATE (decomposition-method) ?psm
  ((has-input-role :value has-schedule-state)
   (has-output-role :value generates-schedule-state)
   (has-schedule-state :type schedule-state)
   (generates-schedule-state :type schedule-state)
   (has-body :value '(lambda (?psm)
                        (in-environment
                         ((?state . (role-value ?psm has-schedule-state))
                          (?schedule-model . (the ?sc (has-schedule-model
                                                         ?state ?sc)))
                          (?constraints . (role-value ?psm has-hard-constraints))
                          (?requirements . (role-value ?psm has-requirements))
                          (?jobs . (role-value ?psm has-jobs))))
                         (if (deadend-state ?state)
                             :nothing
                             (if (requirement-violations ?state ?requirements)
                                 (tell (deadend-state ?state))
                                 (if (constraint-violations ?state ?constraints)
                                     (achieve-generic-subtask
                                      ?psm local-exchange-of-schedule
                                      has-schedule-state ?state)
                                     (if (state-complete ?state ?jobs)
                                         (tell (complete-state ?state))
                                         (achieve-generic-subtask
                                          ?psm generate-new-state-successor
                                          has-schedule-state ?state
                                          has-schedule-context :extend)
                                          (if (constraint-violations
                                              ?state ?constraints)
                                              (achieve-generic-subtask
                                               ?psm exchange-schedule
                                               has-schedule-state ?state))))))))))
  :own-slots ((tackles-task-type schedule-from-state)
              (has-generic-subtasks '(generate-new-state-successor
                                       local-exchange-of-schedule
                                       exchange-schedule))))

```

### 7.3.6.3 Fixing the constraint violations

Task `local-exchange-of-schedule` is invoked to fix the constraint violations locally that occurred while constructing a schedule. This task is achieved by defining a method called `exchange-locally`. The body of this method takes as input a schedule state, say  $S_{s1}$ , which has a number of constraint violations and then it invokes a task `generate-new-state-successor` in the **exchange** context. The body of this task collects all the constraint violations, as the current foci, and then selects the first constraint violation as a focus. Having selected a candidate focus, the assignments of the two conflicting jobs are exchanged. If a schedule state, say  $S_{s2}$ , is reached which has fewer constraint violations then such a schedule state is returned as an output. After each cycle, the task `evaluate-hard-consistency` (cf. Section 6.3.1.1) is invoked to check whether all the constraint violations have been fixed through local exchanges among the job assignments. The following box shows the OCML definitions of `local-exchange-of-schedule` and `exchange-locally`.

```

(def-class LOCAL-EXCHANGE-OF-SCHEDULE (goal-specification-task) ?task
  ((has-input-role :value has-schedule-state)
   (has-output-role :value has-output-state)
   (has-schedule-state :type schedule-state)
   (has-output-state :type schedule-state)
   (has-goal-expression :value (kappa (?task ?s)
                                       (and (schedule-state ?s)
                                             (has-output-state ?task ?s))))))

(def-class EXCHANGE-LOCALLY (primitive-method) ?psm
  ((has-body :value (lambda (?psm)
                      (repeat
                       (in-environment
                        ((?input . (role-value ?psm has-schedule-state))
                         (?output . (achieve-generic-subtask
                                     ?psm generate-new-state-successor
                                     has-schedule-state ?input
                                     has-schedule-context :exchange)))
                        (if (schedule-state ?output)
                            (do (achieve-generic-subtask
                                ?psm evaluate-hard-consistency
                                has-schedule-state ?output
                                has-schedule-context :exchange)
                                (if (< (cardinality
                                       (the ?cs (constraint-violations
                                                ?output ?cs)))
                                      (cardinality
                                       (the ?cs (constraint-violations
                                                ?input ?cs))))
                                (return ?output)))))))))
   :own-slots ((tackles-task-type local-exchange-of-schedule)
                (has-generic-subtasks `(generate-new-state-successor
                                         evaluate-hard-consistency))))

```

If the constraint violations cannot be fixed locally then the task `generate-new-state-successor` is invoked again in the **extend** context and the schedule construction process is resumed. Once a complete schedule is devised then the task `exchange-schedule` is invoked to fix all the outstanding constraint violations through global exchanges. This task is achieved by defining a method called `focus-based-schedule-exchange`. The body of this method is an exhaustive control loop that calls itself until all the constraint violations are fixed. The method collects all the outstanding constraint violations, selects the first constraint violation from this list, retrieves all exchange-operators applicable to fix the selected violation, sorts them, and then applies the first exchange-operator from the sorted list to exchange the assignment of the jobs involved in the constraint violation. After each cycle, relation `schedule-satisfies-constraints` (cf. Appendix 1) is used to check whether all the constraint violations are fixed. The following box shows the OCML definitions of task `exchange-schedule` and method `focus-based-schedule-exchange`.

```

(def-class EXCHANGE-SCHEDULE (goal-specification-task) ?task
  ((has-input-role :value has-schedule-state)
   (has-output-role :value has-output-state)
   (has-schedule-state :type schedule-state)
   (has-output-state :type schedule-state)
   (has-goal-expression :value (kappa (?task ?s)
                                       (and (schedule-state ?s)
                                             (not
                                              (constraint-violations ?s ?any)))))))

(def-class FOCUS-BASED-SCHEDULE-EXCHANGE (primitive-method) ?psm
  ((has-body :value (lambda (?psm)
                      (REPEAT
                       (in-environment
                        ((?input . (role-value ?psm has-schedule-state))
                         (?output . (achieve-generic-subtask
                                     ?psm generate-new-state-successor
                                     has-schedule-state ?input
                                     has-schedule-context :exchange)))
                        (if (schedule-state ?output)
                            (in-environment
                             ((?record . (the-state-search-control-record
                                           ?output))
                              (?focus . (the-slot-value ?record
                                                         'has-schedule-focus))
                              (?sc . (the-slot-value ?output
                                                         has-schedule-model)))
                             (if (schedule-satisfies-constraint ?sc ?focus)
                                 (return ?output))))))))))
  :own-slots ((tackles-task-type exchange-schedule)
              (has-generic-subtasks '(generate-new-state-successor))))

```

#### 7.3.6.4 Foci collection and a focus selection in P&E

As described by the informal schema of the P&E method (cf. Section 7.3.6.1), in the propose phase a job with the lowest degrees of freedom (i.e., a job with the least number of resources and time ranges left for the assignment) is selected as a focus. The default job selection method `job-selected-based-on-lowest-degrees-of-freedom` (cf. Section 6.3.5) from `Generic-Schedule` is used to select a focus, and therefore, no configuration is required to select a focus in the propose phase.

During the exchange phase, first all the constraint violations are collected as foci by using a method called `collect-all-culprit-violations`. This method achieves task `collect-state-foci` (cf. Section 6.3.3) from `Generic-Schedule`. Having collected the foci, the first constraint violation from the list of collected foci is selected as a focus by using a method called `select-the-violation`, which achieves task `select-schedule-focus` (cf. Section 6.3.5) from `Generic-Schedule`.

#### 7.3.6.5 Collection and selection of the exchange operators

All the exchange-operators that can be applied to fix a selected focus are collected by using a method `default-exchange-operator-collection`, which achieves the task `collect-focus-operators` (cf. Section 6.3.6) from `Generic-Schedule`. All the collected exchange-operators are then sorted by using relation `schedule-operator-order` (cf. Section 6.2.2) and the first operator from the sorted list is selected to fix the constraint violation. Having fixed the currently selected focus, the entire problem-solving cycle is repeated to fix the remaining constraint violations and if no more

constraints are violated then a consistent schedule is returned as an output. The following box shows the OCML definition of method `default-exchange-operator-collection`.

```
(def-class DEFAULT-EXCHANGE-OPERATOR-COLLECTION (primitive-method) ?psm
  ((has-body :value (lambda (?psm)
    (setofall ?op
      (and (exchange-operator
        ?op applicable-to-constraints ?l)
        (member (role-value ?psm
          'has-schedule-focus)
          (eval ?l)))))))
  :own-slots ((tackles-task-type '(collect-focus-operators))
    (applicability-condition
      (kappa (?task)
        (= (role-value ?task has-schedule-context) :exchange)))))
```

In total seven new definitions have been defined to model the P&E method by specialising `Generic-Schedule`. Table 7.7 summarises the knowledge requirements of the P&E method.

Table 7.7. The knowledge requirements of the P&E method.

Knowledge Roles	Propose and Exchange
Inference knowledge	<p>The knowledge required to select the operators in both the phases of method</p> <p>A schedule focus selection knowledge in both the phases</p> <p>A schedule state selection knowledge</p> <p>The knowledge required to select the resources and time ranges that can be assigned to jobs in the propose phase</p> <p>The knowledge required to exchange the job assignments involved in conflict</p>
Additional subtasks	<p>Local-exchange-of-schedule,</p> <p>Exchange-schedule, Collect-all-culprit-violations, Default-exchange-operator-collection</p>
Method specific control regime	Propose&Exchange-state
Schedule-context	Extend, exchange
Schedule-focus	Job, constraint violation
Schedule focus selection strategy	<p>A candidate focus in the propose phase of the method is selected by using the method job-selected-based-on-lowest-degrees-of-freedom (cf. Section 6.3.5)</p>

	The first constraint violation from the list of collected foci is selected as a focus in the exchange phase
Schedule operator type	Schedule-extension-operator Exchange-operator
Schedule operator order	It is determined based on a selected focus
Schedule state selection knowledge	Violated constraints: No or minimal Schedule extension: Maximal Cost: Minimum
Global properties	Complete Locally and globally consistent

### 7.3.7 Propose and Genetical-Exchange (P&GE)

This method was proposed by Poeck and Gappa (1993) to solve the assignment task by using the genetic algorithm schema (Goldberg, 1989). As in the case of the P&E method, we modified the original description of the P&GE method to tackle the time element as well as the assignment of jobs to resources.

#### 7.3.7.1 Initial analysis of the method

The P&GE method initially constructs a complete and consistent schedule, and then it tries to optimise a solution. The method uses a notion of optimality based on the minimisation of the constraint violations. The following bullet points analyse the P&GE method.

- The propose phase constructs a complete schedule by assigning jobs to resources and time ranges by applying schedule-extension-operators (cf. Section 6.2.2). The set of a schedule quadruples generated as a complete solution represents an initial population;
- If any of the constraints imposed on a schedule are violated, then they are ignored until a complete schedule is devised. Once a complete schedule is devised then the genetical-exchange phase is invoked to fix these constraint violations. The constraint violations are fixed by invoking the following two tasks: initial-crossover and final-crossover. A new type of schedule modification operator called genetic-operator is defined to fix the constraint violations. This operator is defined as a subclass of schedule-modification-operator (cf. Section 7.2);
- Finally, if no more constraints are violated then a complete and consistent solution is returned as an output.

The following box shows an informal schema of the P&GE method (Poeck and Puppe, 1993).

```

Population: Initial population set generated by any greedy technique
REPEAT
  Select the parents from the initial population
  New solution = initial crossover
  Optimise a new solution for constraint violations = final crossover
  Survival of the fittest solution (i.e., a complete and consistent schedule)

```

### 7.3.7.2 The method specific control regime of P&GE

The method `generation-of-P&GE` defines the method-specific control regime of P&GE, which is constructed by modifying the method-specific control regime of `Generic-Schedule` (cf. Section 6.3.2) to fix the constraint violations. This control regime first invokes the task `generate-new-state-successor` in the **extend** context to devise a complete schedule. In line with P&R (cf. Section 7.3.4), if the constraints are violated while constructing a schedule then they are ignored until a complete schedule is constructed. Once a complete schedule is constructed, then the task `initial-crossover` is invoked to fix the constraint violations by exchanging *randomly* the conflicting jobs so that a schedule with none or fewer constraint violations is generated.

If the constraint violations cannot be fixed by applying the task `initial-crossover` then the task `final-crossover` is invoked, which applies a more exhaustive strategy to fix the constraint violations. This task takes as input a partially corrected set of assignments (i.e., a population), which are generated by the task `initial-crossover` and then it iteratively exchanges the job assignments involved in a conflict to construct a consistent schedule. The following box shows the OCML definition of `generation-of-P&GE`.

```

(def-class GENERATION-OF-P&GE (decomposition-method) ?psm
  ((has-input-role :value has-schedule-state)
   (has-output-role :value generates-schedule-state)
   (has-schedule-state :type schedule-state)
   (generates-schedule-state :type schedule-state)
   (has-body :value (lambda (?psm)
                       (in-environment
                        ((?state . (role-value ?psm has-schedule-state))
                         (?schedule-model . (the ?sc (has-schedule-model
                                                         ?state ?sc)))
                         (?constraints . (role-value ?psm has-hard-constraints))
                         (?requirements . (role-value ?psm has-requirements))
                         (?jobs . (role-value ?psm has-jobs)))
                        (if (deadend-state ?state)
                            :nothing
                            (if (requirement-violations ?state ?requirements)
                                (tell (deadend-state ?state))
                                (if (state-complete ?state ?jobs)
                                    (achieve-generic-subtask
                                     ?psm generate-new-state-successor
                                     has-schedule-state ?state
                                     has-schedule-context :extend)
                                    (if (constraint-violations ?state ?constraints)
                                        (achieve-generic-subtask
                                         ?psm initial-crossover
                                         has-schedule-state ?state)
                                        (if (constraint-violations ?state constraints)
                                            (achieve-generic-subtask
                                             ?psm final-crossover
                                             has-schedule-state ?state))))))))))
   :own-slots ((tackles-task-type schedule-from-state)
                (has-generic-subtasks '(generate-new-state-successor
                                         initial-crossover final-crossover))))

```

### 7.3.7.3 Fixing the constraint violations in the genetical-exchange phase

The task initial-crossover is achieved by using a method called default-initial-crossover. This method takes as an input a schedule state, which has a number of constraint violations and then it first collects all the constraint violations as the candidate foci. The first constraint violation from the list of collected foci is selected as a focus by using the task select-schedule-focus. Having selected a candidate focus, all genetic-operators that can be applied to fix the selected focus are collected and then sorted to determine the order of their application. The operator collection and sorting operations are performed by invoking the tasks collect-focus-operators and sort-focus-operators respectively. After each cycle, the task evaluate-hard-consistency (cf. Section 6.3.1.1) is invoked to check whether an offspring (i.e., a set of new assignments) has fewer constraint violations than the initial set. If all the constraint violations are fixed by using the task initial-crossover then this consistent schedule state is returned as an output, otherwise the task final-crossover is invoked. The task final-crossover is achieved by using a method called default-crossover. This method takes as input a partially corrected set of assignments generated by the task initial-crossover and then performs a focus-based exchange among the assignments of the jobs that are involved in conflict. The method default-crossover is similar in spirit to the method focus-based-schedule-exchange (cf. Section 7.3.6.4). Having fixed all the constraint violations, the body of method default-

crossover invokes a new task called `evaluate-fitness-function`, which is used to check the schedule quality. A schedule quality is evaluated based on the amount of time by which the jobs fails to meet their deadlines in a final solution. In scheduling, the evaluation function is usually constructed by using optimisation criteria, such as *job tardiness*, *maximisation of the resource utilisation*, or *work-in-progress* (Davis, 1985; Bagchi *et al.*, 1991; Starkweather *et al.*, 1993). In line with these proposals, in our approach the evaluation function is constructed to check the job tardiness in a final solution. The job tardiness is calculated by using the following equation shown in Figure 7.1.

$$\left[ \sum_{j=i}^n j\text{tardi} = (0.1 / \text{maximum lateness}) * 100 \right]$$

**Figure 7.1.** The evaluation-function used to calculate the job tardiness.

The ‘maximum lateness’ in the above equation indicates the time by which a particular job is delayed in a schedule. A tardiness of a job, say  $j_i$ , can be represented by the time by which a job  $j_i$  fails to meet its due date, or otherwise it is considered as zero. The lateness function is represented by the time by which the latest end time of a job,  $j_i$  exceeds its due date and it is represented as follows:  $L_i = \text{Let}_{j_i} - \text{Dd}_{j_i}$ , where  $\text{Let}_{j_i}$  and  $\text{Dd}_{j_i}$  represents the latest end time and the due date of a job  $j_i$  respectively (Baker, 1974). The following box shows the OCML definitions of task `final-crossover`, method `default-crossover`, and task `evaluate-fitness-function`.

```

(def-class FINAL-CROSSOVER (goal-specification-task) ?task
  ((has-input-role :value has-schedule-state)
   (has-output-role :value has-output-state)
   (has-schedule-state :type schedule-state)
   (has-output-state :type schedule-state)
   (has-goal-expression :value (kappa (?task ?s)
                                       (and (schedule-state ?s)
                                             (not (constraint-violations
                                                    ?s ?any)))))))

(def-class DEFAULT-CROSSOVER (primitive-method) ?psm
  ((has-body :value (lambda (?psm)
                      (REPEAT
                       (in-environment
                        ((?input . (role-value ?psm has-schedule-state))
                         (?output . (achieve-generic-subtask
                                     ?psm generate-new-state-successor
                                     has-schedule-state ?input
                                     has-schedule-context :genetical-exchange)))
                       (if (schedule-state ?output)
                           (in-environment
                            ((?record . (the-state-search-control-record
                                           ?output))
                             (?focus . (the-slot-value
                                           ?record 'has-schedule-focus))
                             (?sc . (the-slot-value ?output
                                                    'has-schedule-model)))
                            (if (schedule-satisfies-constraint ?sc ?focus)
                                (return ?output)
                                (do
                                 (achieve-generic-subtask
                                  ?psm evaluate-fitness-function
                                  has-schedule-state ?output)))))))
                      :own-slots ((tackles-task-type final-crossover)
                                   (has-generic-subtasks '(generate-new-state-successor
                                                            evaluate-fitness-function))))))

(def-class EVALUATE-FITNESS-FUNCTION (primitive-task) ?task
  ((has-input-role :value has-output-state)
   (has-output-state :type schedule-state)
   (has-body :value (lambda (?task)
                      ((?output . (role-value ?task has-output-state))
                       (?schedule-model . (the ?sc (has-schedule-model
                                                       ?output ?sc)))
                       (?jobs . (role-value ?task has-jobs)))
                      (if (fitness-function-for-tardiness ?output ?jobs)
                          (tell (state-not-tardy ?output)))))))

```

#### 7.3.7.4 Foci collection and a focus selection in P&GE

All the constraint violations in the genetical-exchange phase are collected as the foci by using a method called `collect-all-violations`. This method achieves the task `collect-state-foci` (cf. Section 6.3.3) from `Generic-Schedule`.

Having collected all the foci, the first constraint violation from the list of collected foci is selected as a candidate focus by using a method called `select-candidate-constraint`.

#### 7.3.7.5 The operator collection and selection in P&GE

Once a candidate focus is selected then all the genetic-operators are collected by using a method called `default-genetical-operator-collection`, which achieves the task `collect-focus-operators` (cf. Section 6.3.6) from `Generic-Schedule`. All the collected genetic-operators are then sorted by using the relation `schedule-operator-order` (cf. Section 6.2.2) to determine the order of their application and the first operator from the sorted list is selected and applied to fix the constraint violation. The

following box shows the OCML definition of method default-genetical-operator-collection.

```
(def-class DEFAULT-GENETICAL-OPERATOR-COLLECTION (primitive-method) ?psm
  ((has-body :value (lambda (?psm)
    (setofall ?op
      (and (genetic-operator
        ?op applicable-to-constraints ?l)
        (member (role-value
          ?psm 'has-schedule-focus)
          (eval ?l)))))))
  :own-slots ((tackles-task-type '(collect-focus-operators))
    (applicability-condition
      (kappa (?task)
        (and (= :genetical-exchange
          (role-value ?task has-schedule-context))
          (genetically-exchangeable
            (role-value ?task 'has-schedule-focus)))))))
```

In total eight new definitions are defined in order to model the P&GE method by specialising Generic-Schedule. Table 7.8 summarises the knowledge requirements of P&GE.

Table 7.8. The knowledge requirements of the P&GE method.

Knowledge Roles	Propose and Genetical-Exchange
Inference knowledge	<p>The operator selection knowledge in both the phases</p> <p>The schedule state selection knowledge</p> <p>The focus selection knowledge in both the phases</p> <p>In the propose phase the knowledge required to assign resources and time ranges to jobs</p> <p>In the genetical-exchange phase the knowledge required to fix the constraint violations and to optimise a consistent schedule in terms of constraint violations</p>
Additional subtasks	<p>Initial-crossover, Default-initial-crossover, Final-crossover, Default-crossover, Evaluate-fitness-function, Collect-all-violations, Select-candidate-constraint, Default-genetical-operator-collection</p>
Method specific control regime	Generation-of-P&GE
Schedule-context	Extend, genetical-exchange

Schedule-focus	Job, constraint violation
Schedule focus selection strategy	In the propose phase one of the job selection methods (cf. Section 6.3.5) from Generic-Schedule is used to select a job  In the genetical-exchange phase the first constraint violation from the list of collected foci is selected
Schedule operator type	Schedule-extension-operator Genetic-operator
Schedule operator order	It is determined based on a selected focus
Schedule state selection knowledge	Violated constraints: No Schedule extension: Maximal Cost: Minimum
Global properties	Complete, consistent, and globally optimal (optimality is considered by minimising the number of constraint violations)

With the description of the P&GE method, we conclude our presentation of all the PSMs in our library. In the following section we describe how these PSMs can be categorised.

## 7.4 Categorisation of the methods

All the PSMs in our library can be categorised based on the way they cover and solve the different types of schedules. For instance, the P&R method can be used to devise a complete schedule and fix the constraint violations, which occur while constructing a schedule. The following bullet points describe the categorisation of the methods from our library:

- **Schedule completeness:** The methods from this category are constructive in nature because they can be used to construct a complete solution schedule. To devise a complete solution schedule, these methods select a schedule state that does not violate any constraints or requirements, and when encountered with an inconsistent schedule state such a schedule state is either ignored or the search backtracks to the last consistent schedule state to resume the schedule construction process. Generic-Schedule and P&B methods fall into this category.
- **Schedule consistency and feasibility:** The methods from this category can be used to repair different types of inconsistencies, such as constraint or requirement violations that occur while constructing a schedule. In contrast with the methods that deal with

schedule completeness, the schedule state selection policy of these methods takes into account all those schedule states that violate constraints or requirements such that these inconsistencies can be fixed by applying the methods. The P&R, P&Rf, P&E, and P&GE methods fall into this category.

- **Schedule optimisation:** The methods from this category try to optimise a complete schedule. The hill climbing and P&I methods from our library fall into this category.

Figure 7.2 depicts the categorisation of the methods as discussed in the above bullet points.

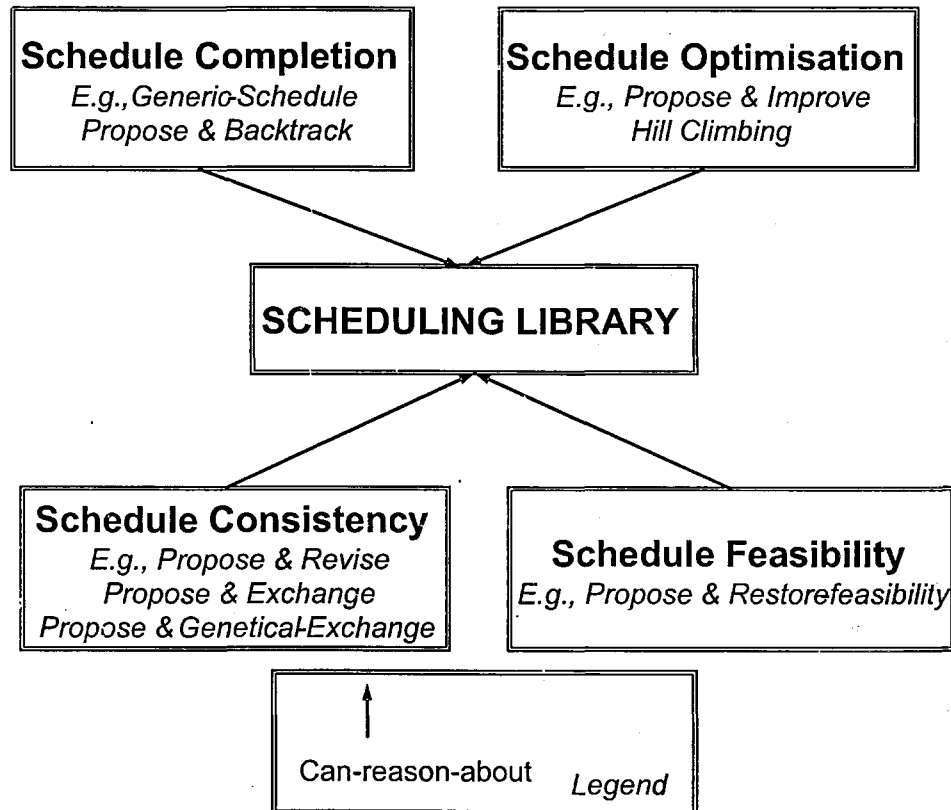


Figure 7.2. Categorisation of the methods in the library.

## 7.5 Conclusion

In this chapter we have described how different PSMs in our library have been constructed uniformly by reusing *Generic-Schedule*. All the PSMs in our library have been defined by reusing or specialising schedule state selection knowledge, operator construction and selection knowledge, method-specific control regime, and the notions of context and focus. This uniform approach allows us to compare and contrast the knowledge requirements of these PSMs. On average, less than a dozen definitions were required to be defined to engineer a new PSM. In contrast with existing proposals (Hori and Yoshida, 1998; Sundin, 1994; Le Pape, 1994; Tijerino and Mizoguchi, 1993), our library provides a comprehensive coverage to solve the different schedule types. Moreover, in contrast with some of the

existing proposals (Hori and Yoshida, 1998), because our library does not subscribe to any particular application or scheduling domain it has a wider applicability.

In the next chapter, we describe the validation of our library which has been carried out to confirm the generic nature of our library.

# Chapter 8

## EVALUATION STUDY OF THE LIBRARY

In this chapter, we describe the validation of our library carried out on a number of scheduling applications to confirm its generic nature and its applicability to real-world problems. In particular, we validate the following claims: a) the overall framework provides appropriate distinctions necessary to support rapid KBS development by reuse, b) the scheduling task ontology can be effectively used to characterise the different types of scheduling problems, and c) different methods in the library can be effectively applied to construct scheduling applications. Our library has been validated on five scheduling domains: satellite-scheduling, the CIPHER project schedule application, daily ship-maintenance, weekly ship-maintenance, and a benchmark application used in the scheduling area. Our evaluation study helps in validating the static and dynamic properties of KBS (Preece *et al.*, 1996) and in doing so it validates our library framework. Table 8.1 describe the different types of scheduling problem types, solution criteria covered by the applications, and at the same time it also states the nature of the application data, i.e. whether it is from a real-life, a non real-life, or a benchmark.

Table 8.1. Properties of the scheduling applications.

Application name	Scheduling problem types	Schedule solution criteria	Nature of the application data
The satellite-scheduling application	Space scheduling	Complete, optimal	Non real-life
CIPHER- a resource allocation application	Resource allocation problem	Complete	Real-life
The daily ship-maintenance application	Joint scheduling problem	Complete, feasible (no requirement violation)	Real-life
The weekly ship-maintenance application	Joint scheduling problem	Complete, consistent (no constraint violation)	Real-life
The benchmark application	Variant of the job-shop scheduling problem	Complete	Benchmark data

## 8.1 The satellite-scheduling application

The satellite-scheduling application can be characterised as the assignment of satellites to the available antennas to ensure earth-satellite communication at different times during a 24 hr. scheduling horizon. The satellite-scheduling application is rather complex due to its dynamic environment, non-monotonic nature of the various constraints, and the varying degrees of satellite-antenna communication patterns. It is crucial that all the constraints and requirements must be maintained at all times while devising a schedule for this application.

### 8.1.1 Construction of a task model

In accordance with the task ontology, satellites are represented as *jobs* and antennas as limited supply *resources* to which satellites can be assigned to perform communications. The communications within each satellite are represented as *activities*. The following bullet points describe the satellite-scheduling application in further detail.

- The application comprises of five satellites: Nimbus-1, Nimbus-2, Chandra-1, Meteorological-1, and Meteorological-2. Each satellite requires four 15 minute long communication activities that need to be performed within a specific time range;
- There are three antennas: Low-Range-Antenna, Wide-Range-Antenna, and Meteorological-Antenna. Each antenna has a fixed capacity that represents the total number of satellites it can handle at any given time. For instance, Low-Range-Antenna and Wide-Range-Antenna can handle two satellites at any given time, whereas Meteorological-Antenna can only handle one at a time. Each antenna also has a limited visibility period and therefore it can communicate with a specific satellite only at certain times.

The notion of a satellite and an antenna is formalised by defining application-specific classes, such as *satellite-job* and *antenna-resource*. These classes are defined as subclasses of class *job* (cf. Section 5.2.2) and *resource* (cf. Section 5.2.3) respectively. The following box shows the OCML definitions of the Nimbus-1 satellite and Low-Range-Antenna. The representation of other satellites and antennas can be realised in the same way.

```

(def-class SATELLITE-JOB (job))

(def-class NIMBUS-1-JOB (satellite-job))

(def-class NIMBUS-1-JOB-TIME-RANGE (job-time-range))

(def-instance NIMBUS-1 nimbus-1-job
  ((requires-resource '(low-range-antenna))
   (has-activities '(nimbus-1-communication-1 nimbus-1-communication-2
                    nimbus-1-communication-2 nimbus-1-communication-4))
   (has-time-range nimbus-1-time-range)
   (has-duration 60-minute-duration)))

(def-class ANTENNA-RESOURCE (resource))

(def-class LOW-RANGE-ANTENNA-RESOURCE (antenna-resource))

(def-instance LOW-RANGE-ANTENNA low-range-antenna-resource
  ((has-job-belonging nimbus-1)
   (has-availability generic-antenna-time-range)
   (has-capacity 2)))

(def-instance NIMBUS-1-TIME-RANGE nimbus-1-job-time-range
  ((has-earliest-start-time (new-instance 'time-point '((hour-of 00)
                                                         (minute-of 00))))
   (has-latest-end-time (new-instance 'time-point '((hour-of 09)
                                                     (minute-of 00))))))

(def-instance GENERIC-ANTENNA-TIME-RANGE time-range
  ((has-start-time (new-instance 'time-point '((hour-of 15)
                                                (minute-of 01))))
   (has-end-time (new-instance 'time-point '((hour-of 13)
                                              (minute-of 59))))))

```

In the following section we describe the different constraints and requirements associated with the satellite-scheduling application.

### 8.1.2 Modelling constraints and requirements

The satellite-scheduling application is formulated based on the following constraints and requirements.

1. **Antenna-visibility-constraint:** Each antenna has a fixed limited visibility period during which all the communication activities of the satellites must be completed. A set of five antenna visibility constraints are defined to impose this constraint between satellites and their respective antennas. The following box shows the OCML definition of one such antenna visibility constraint imposed between the Nimbus-1 satellite and Low-Range-Antenna. The antenna-visibility-constraint for other satellites and antennas can be realised analogously.

```

(def-class ANTENNA-VISIBILITY-CONSTRAINT (constraint))

(def-instance NIMBUS-1-TO-LOW-RANGE-ANTENNA antenna-visibility-constraint
  ((applicable-to-jobs '(setofall ?x (nimbus-1-job ?x)))
   (has-expression
    (kappa (?sc)
      (forall (?a ?jtr)
        (=> (and (member (nimbus-1 ?a low-range-antenna ?nimbus-1-job-
time-range) ?sc)
                  (not
                   (TIME-RANGES-INTERSECT
                    '?nimbus-1-job-time-range
                    no-nimbus-1-to-low-range-antenna-visibility))))))))))

```

Nimbus-1-to-low-range-antenna constraint states that the Nimbus-1 satellite cannot communicate with Low-Range-Antenna between the time-window of 12:31 to 23:59, which is the non-visibility period of Low-Range-Antenna; otherwise an inconsistency is reported;

2. **Communication-duration:** This constraint is again common to all the satellites, which states that each communication slot within each satellite must be of 15 minutes duration. In total, a set of five constraints are defined to impose the communication duration constraint on the five satellites;
3. **Number-of-communication-slots:** This requirement is common to all the satellites and states that each satellite must have four communication slots per day with its antenna. The following box shows the OCML definition of this requirement.

```

(def-class SATELLITE-JOB-REQUIREMENT (requirement))

(def-instance FOUR-COMMUNICATIONS-PER-DAY-PER-SATELLITE
  satellite-job-requirement
  ((applicable-to-jobs '(setofall ?x (satellite-job ?x)))
   (has-expression (kappa (?sc)
      (exists ?x
        (and (satellite-job ?x)
              (has-activities
               ?x ?satellite-communication)
              (= (number-of-activities-within-job
                  ?x) 4)))))))

```

As it can be seen in the above box, in order to check whether all the satellites have four communication slots we used the function `number-of-activities-within-job` (cf. Appendix 1), which retrieved all the communication activities associated with each satellite and then an equality condition is imposed stating that number of communication activities of satellites must be equal to four;

4. **Communication-gaps:** This requirement is also common to all the satellites and states that the gap between any two communication slots within each satellite should not be greater than five hours. The following box shows the OCML definition of this requirement.

```

(def-instance NO-COMMUNICATION-GAPS-GREATER-THAN-FIVE-HOURS
  satellite-job-requirement
  ((applicable-to-jobs '(setofall ?x (satellite-job ?x)))
   (has-expression
    (kappa (?sc)
     (exists ?x
      (and (has-activities ?x ?list)
           (exists ?a1
            (and (satellite-communication ?a1)
                 (member ?list ?a1)
                 (has-time-range ?a1 ?jtr)
                 (has-earliest-start-time ?jtr ?t1)
                 (exists ?a2
                  (and (satellite-communication ?a2)
                       (member ?list ?a2)
                       (has-time-range ?a2 ?jtr2)
                       (has-earliest-start-time
                        ?jtr2 ?t2)
                       (duration-is-less-than
                        (time-entity-difference
                         '?t2 '?t1)
                        (5 hour))))))))))))))

```

In order to check whether the communication gap between any two communication activities, say  $C_1$  and  $C_2$ , associated with the same satellite is less than five hours, we first calculated the time entity difference between the earliest start times of  $C_1$  and  $C_2$ , and then a relation called `duration-is-less-than` is used to state that the effective time difference between time points  $t_2$  and  $t_1$  (which represents the earliest start time of  $C_2$  and  $C_1$  respectively) is less than five hours.

In summary, we did not encounter any particular problem while formalising the satellite-scheduling application. Only a few additional application-specific relations and functions were defined to model constraints and requirements. More importantly, the key classes from the task ontology, such as `job`, `resource`, `activity`, and `job-time-range` have provided an adequate level of detail to capture the application-specific knowledge precisely. In a nutshell, our task ontology has provided an adequate modelling leverage to formalise the satellite-scheduling problem. In the following section we describe how a schedule for this application was constructed by using the different PSMs from our library.

### 8.1.3 Devising a complete schedule by using Propose & Backtrack

The main solution requirement for this application is to construct a complete schedule, and therefore, we first used the Propose & Backtrack (P&B) method from the library. In the following section we describe how P&B was configured to tackle this application.

#### 8.1.3.1 Construction of the operators

The P&B method can be configured by defining the following two types of application-specific operators, which are used to assign satellites to antennas and to their respective time ranges: `satellite-schedule-operator` and `satellite-schedule-time-range-operator`. `Satellite-schedule-operator` is defined by complying with the 'number-of-communication-slots' requirement. In other words while assigning the

satellites, `satellite-schedule-operator` makes sure that each satellite has four communication slots with its assigned antennas. `Satellite-schedule-time-range-operator` is defined in such a way that a correct start and end time is assigned to each satellite. The following box shows the OCML definitions of these operators defined for the `Nimbus-1` satellite.

```
(def-class SATELLITE-SCHEDULE-OPERATOR (schedule-extension-resource-operator))

(def-class SATELLITE-SCHEDULE-TIME-RANGE-OPERATOR
  (schedule-extension-time-range-operator))

(def-instance NIMBUS-1-TO-LOW-RANGE-ANTENNA satellite-schedule-operator
  ((applicable-to-jobs '(setofall ?x (nimbus-1-job ?x)))
   (has-costs 6)
   (has-body (lambda (?x ?sc)
                (the ?low-range-antenna-resource
                  (and (handles-job ?low-range-antenna-resource ?x)
                       (has-activities ?x nimbus-1-comm)
                       (= (length ?nimbus-1-comm) 4)))))))

(def-instance NIMBUS-1-TO-TIME-RANGE satellite-schedule-time-range-operator
  ((applicable-to-jobs '(setofall ?x (nimbus-1-job ?x)))
   (has-costs 6)
   (has-body (lambda (?x ?sc)
                (the ?nimbus-1-job-time-range
                  (and (schedule-model ?sc)
                       (nimbus-1-job-time-range ?nimbus-1-job-time-range)))))))
```

Each operator also has a specific cost associated with it, which represents the cost incurred by the assignment of each satellite. To represent the cost of each satellite assignment, a new slot called `has-costs` is added to `schedule-extension-resource-operator` and `schedule-extension-time-range-operator` (cf. Section 6.2.2). A new function called `satellite-state-cost-function` is defined in order to calculate the cost of a satellite schedule. The following box shows the OCML definition of `satellite-state-cost-function`.

```
(def-function SATELLITE-STATE-COST-FUNCTION (?sc)
  :body (in-environment
        ((?input-state . (the ?input-state
                              (schedule-state ?input-state
                                has-schedule-model ?sc))))
        (if (state-transition ?s1 ?op ?input-state)
            (get-sum-of-all-cost
             (satellite-state-cost-function
              (the ?s1-sc
                  (has-schedule-model ?s1 ?s1-sc)))
              (the ?c (has-costs ?op ?c)))
            '(00000000))))

(def-function GET-SUM-OF-ALL-COST (?vect ?op-cost)
  :body (in-environment
        ((?v-pos . (- 9 ?op-cost)))
        (if (= ?op-cost 0)
            ?vect
            (append (sublist ?vect ?v-pos)
                     (list-of (+ 1 (elt ?v-pos ?vect))
                              (nthrest ?vect (+ 1 ?v-pos)))))))
```

Having defined all the operators, the relation `schedule-operator-order` (cf. Section 6.2.2) is instantiated to determine the order in which these operators are applied to assign

satellites to antennas and time ranges. The following box shows how the order of operators for assigning Nimbus-1 satellite is determined.

```
(tell (schedule-operator-order nimbus-1-to-low-range-antenna
                               nimbus-1-to-time-range))
```

### 8.1.3.2 Focus selection knowledge and schedule construction

Because the satellite-scheduling application did not impose any particular condition for selecting a correct focus, the focus in this application is selected by using the default focus selection method `job-selection-based-on-lowest-degrees-of-freedom` (cf. Section 6.3.5). This method subscribes to the DSR heuristic (Dechter and Meiri, 1989), which in this case ensures that a satellite with the least number of antennas left for its assignment is selected as a candidate focus. Figure 8.1 shows the order in which satellites are selected for their assignment.

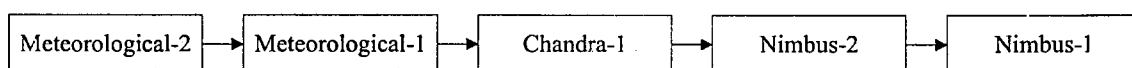


Figure 8.1. The order in which satellites are selected for their assignment.

By determining how a correct focus can be selected, the configuration of P&B is completed and then the satellite-scheduling application is executed to construct a complete schedule. Hence little configuration effort is required to configure the P&B method in order to tackle the satellite-scheduling application. Only one new slot, `has-costs` was added to the definition of operators to represent the cost of satellite assignments, while two application-specific functions were defined to calculate the cost of an assignment. The complete schedule for the satellite-schedule application was constructed by generating 464 schedule states. Moreover, thanks to the correct focus selection knowledge no backtracking was required, and therefore, 100% efficiency was achieved during schedule construction. The function, `satellite-state-cost-function` calculated the total cost of a schedule, which was represented in terms of a 9-place vector. Initially, no satellites were assigned and therefore the cost of the empty satellite schedule was 9-place vector with all zeros, subsequently each time a new satellite was assigned, the cost of the satellite schedule was calculated by adding the cost associated with the currently assigned satellite to the cost of the previously assigned satellites. The total cost of a complete schedule was (000000120).

### 8.1.4 Trying to optimise a complete schedule by Hill climbing

Although, the schedule generated by the P&B method was of a ‘good’ quality (i.e., a solution did not violate any constraints and maintained all the requirements), it was not an optimal one. The main reason for this is that the assignment of the last two satellites, i.e.

Nimbus-2 and Nimbus-1, were leading towards a state with the same cost. In order to find an optimal solution we used the hill climbing method. As in the case of the P&B method, the hill climbing method also failed to find an optimal solution, because this method does not have enough discriminating knowledge to break the tie between the costs of these two assignments. Moreover, the hill climbing method not only failed to devise an optimal solution but it also reduced the overall efficiency of a schedule construction by 20% in comparison with the P&B method. The primary reason for this is that the state selection policy of the hill climbing method generates all the possible successor schedule states of a current schedule state before selecting the locally best state, whereas the P&B method generates only a single successor schedule state.

#### 8.1.5 Optimising a complete schedule by P&I

Because the previous two methods failed to devise an optimal solution, we finally applied the P&I method from our library. The propose phase of the method is configured straightforwardly from *Generic-Schedule* without any further refinement and therefore here we focus on describing how we configured the improve phase of the method.

The basic idea of the P&I method is to construct a complete solution quickly and then in a second stage the improve phase chooses the best possible improvement to the solution. The improve phase is configured by defining a new type of application-specific improvement operator called *satellite-schedule-improvement-operator*. This operator is used to break the tie between the assignment of Nimbus-1 and Nimbus-2 satellites. In particular, because the assignments of these two satellites are competing with each other we decided to change the order of their assignment by swapping their time range assignments. Essentially the swapping of the time range assignment changed the order in which these two satellites performed their communication. This change in position effectively optimised the ‘locking period performance’<sup>1</sup> between a satellite and its respective antenna.

##### 8.1.5.1 Foci collection and focus selection in the improve phase

Once a complete schedule is constructed by using the propose phase, all the *assigned satellites* are collected as foci as they are potential candidates for improvement, and a satellite with the highest cost of assignment is selected as the candidate focus. Because the assignment of Nimbus-1 has a higher cost over Nimbus-2, it is selected as a focus.

---

<sup>1</sup> The ‘locking period performance’ represents the time-span of each satellite to establish a communication with antennas. By optimising this period it helps to reduce the cost spent on carrying out the communication activities.

Having completed the configuration of the P&I method, the satellite-scheduling application is executed. As it can be seen in the following box the cost of a solution generated by swapping Nimbus-1 and Nimbus-2 satellites is now (000000108). In other words, a 10% improvement in the cost of a complete solution is achieved simply by swapping the time range assignments of these two satellites. The following box shows the synoptic trace of the behaviour of the P&I method for the satellite-scheduling application. An optimal schedule is constructed by generating 512 schedule states.

```

----- Enter task EVALUATE-COMPLETENESS515 with arguments (HAS-SCHEDULE-
STATE SCHEDULE-STATE509)

----- Exit task EVALUATE-COMPLETENESS515 -> (STATE-COMplete SCHEDULE-
STATE509)

----- Enter task DEFAULT-COST-EVALATION517 with arguments (HAS-SCHEDULE-
STATE SCHEDULE-STATE509)

----- Exit task DEFAULT-COST-EVALATION517 -> (000000120)

OCML 16 : 1 > (describe-instance 'schedule-state509)
Instance SCHEDULE-STATE509 of class SCHEDULE-STATE
HAS-SCHEDULE-MODEL: ((NIMBUS-1 LOW-RANGE-ANTENNA NIMBUS-1-COMMUNICATION-4 NIMBUS-1-TIME-
RANGE) (NIMBUS-2 WIDE-RANGE-ANTENNA NIMBUS-2-COMMUNICATION-4 NIMBUS-2-TIME-RANGE)
(CHANDRA-1 WIDE-RANGE-ANTENNA CHANDRA-1-COMMUNICATION-4 CHANDRA-1-TIME-RANGE)
(METEOROLOGICAL-1 METEOROLOGICAL-ANTENNA METEOROLOGICAL-1-COMMUNICATION-4 METEOROLOGICAL-
1-TIME-RANGE) (METEOROLOGICAL-2 LOW-RANGE-ANTENNA METEOROLOGICAL-2-COMMUNICATION-4
METEOROLOGICAL-2-TIME-RANGE))

We improve a schedule from here.....

----- Enter task PROPOSE-AND-IMPROVE-STATE524 with arguments (HAS-SCHEDULE-SPACE
SCHEDULE-SPACE287) (HAS-SCHEDULE-STATE SCHEDULE-STATE512)

----- Enter task GENERATE-NEW-STATE-SUCCESSOR526 with arguments (HAS-SCHEDULE-
STATE SCHEDULE-STATE512) (HAS-SCHEDULE-CONTEXT :IMPROVE)

----- Enter task COLLECT-IMPROVABLE-JOBS528 with arguments (HAS-SCHEDULE-STATE
SCHEDULE-STATE512) (HAS-SCHEDULE-CONTEXT :IMPROVE)

----- Exit task COLLECT-IMPROVABLE-JOBS528 -> (NIMBUS-1 NIMBUS-2 CHANDRA-1
METEOROLOGICAL-1 METEOROLOGICAL-2)

----- Enter task PROPOSE-SCHEDULE-FROM-CONTEXT530 with arguments
(HAS-SCHEDULE-STATE SCHEDULE-STATE512) (HAS-SCHEDULE-CONTEXT :IMPROVE)

----- Enter task SELECT-MOST-EXPENSIVE-JOB532 with arguments (HAS-SCHEDULE-FOCI
(NIMBUS-2 NIMBUS-1 CHANDRA-1 METEOROLOGICAL-1 METEOROLOGICAL-2))

----- Exit task SELECT-MOST-EXPENSIVE-JOB532 -> NIMBUS-1

The cost of a schedule after swapping the assignments of satellites

----- Enter task DEFAULT-COST-EVALATION558 with arguments (HAS-SCHEDULE-
STATE SCHEDULE-STATE512)

----- Exit task DEFAULT-COST-EVALATION558 -> (000000108)

```

Figure 8.2 shows a comparison between the average CPU times required to assign each satellite by using all the three methods. It can be observed in the following graph that the assignment of the Nimbus-1 and Nimbus-2 satellites consumed the maximum CPU time while using all the three methods mainly because the assignment of these two satellites are competing with each other throughout the schedule construction.

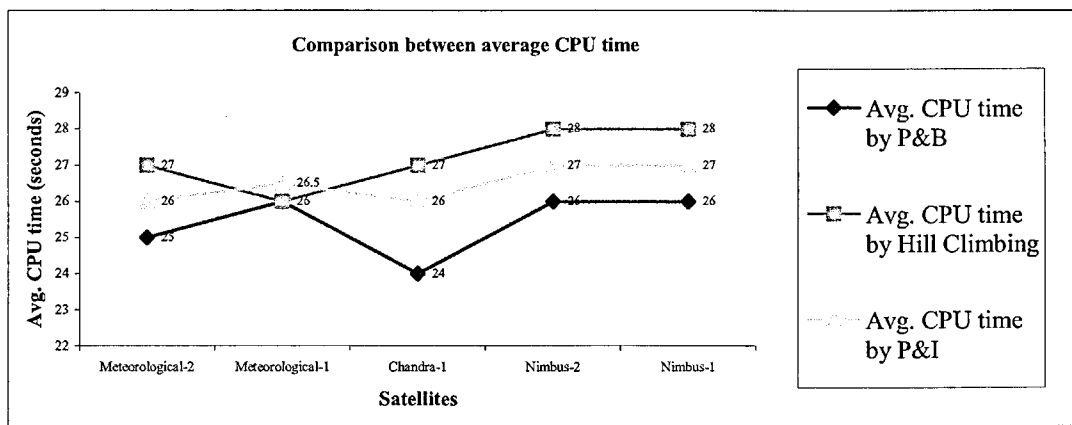


Figure 8.2. Comparison between the average CPU times required to assign the satellites.

## 8.2 CIPHER – a resource allocation application

CIPHER is a real-life collaborative project among six academic and industrial partners. To maintain the anonymity of the academic and industrial organisations involved in the project, we will refer to them as co-ordinator-1, contractor-2, contractor-3, contractor-4, contractor-5, and contractor-6. The project comprises twelve work-packages, and each of them includes a number of tasks, which must be achieved in order to complete the work-package. Each organisation has a limited number of people available to carry out the work prescribed by the various work-packages, and therefore, all project members are treated as limited capacity resources. The goal of the CIPHER application is to construct a complete schedule by allocating all the work-packages and related tasks to the available project members in accordance with a number of constraints.

### 8.2.1 Construction of a task model

In accordance with the task ontology work-packages are treated as jobs, project members as resources, and all the tasks within work-packages as activities. The schedule is constructed for a period of 30 months.

Each project member is assumed to be a unary capacitated resource. This capacity constraint must be maintained at all time in the scheduling horizon. Moreover, the total capacity of each organisation must be distributed among relevant work packages in accordance with the distribution given in the project specification. For instance, co-ordinator-1 includes the following three people: co-ordinator-1-person-1, co-ordinator-1-person-2, and co-ordinator-1-person-3, for a total capacity of 75 person-months. Obviously given that CIPHER is a 2 ½ years project no staff member can provide more than 30 person-months to the project. Therefore, the total capacity of 75 person-months of co-ordinator-1 is distributed as follows:

Co-ordinator-1-person-1 = 30 person-months, Co-ordinator-1-person-2 = 30 person-months, Co-ordinator-1-person-3 = 15 person-months.

A set of 30 instances are associated with co-ordinator-1-person-1 and co-ordinator-1-person-2 to represent their capacity of 30 person-months while 15 instances are defined to represent the capacity of co-ordinator-1-person-3. The capacity distribution of all project members is shown in Table 8.2. In total 330 instances were defined to represent the total capacity of all project members.

Table 8.2. The capacity distribution of all the project-staffs.

Name of the partner = total capacity	Person-1-capacity (person-months)	Person-2-capacity (person-months)	Person-3-capacity (person-months)
Co-ordinator-1 = 75	30	30	15
Contractor-2 = 60	30	30	-
Contractor-3 = 60	30	30	-
Contractor-4 = 30	30	-	-
Contractor-5 = 75	30	30	15
Contractor-6 = 30	30	-	-

Each work-package in CIPHER also has a specific requirement for the total number of project-months required for its completion. This requirement must be taken into account while assigning the work-packages. Table 8.3 shows the project-staff requirement of all the twelve work-packages. In Table 8.3, the abbreviations CO and CR represent co-ordinator and contractor respectively.

Table 8.3. The resource requirement of each work-package.

Project-staffs Work-packages	CO-1	CR-2	CR-3	CR-4	CR-5	CR-6
Work-package-1	15	2	2	1	2	1
Work-package-2	7	13	7	3	6	2
Work-package-3	6	3	18	-	6	-
Work-package-4	6	4	3	-	12	-
Work-package-5	6	4	3	-	14	-
Wprk-package-6	10	4	3	1	5	-
Work-package-7	2	14	2	1	3	-
Work-package-8	6	6	6	7	5	5

Work-package-9	6	6	6	7	7	5
Work-package-10	7	7	7	7	8	5
Work-package-11	2	1	1	1	1	7
Work-package-12	2	3	3	2	3	3

The notion of a work-package is represented by defining the application-specific class called `cipher-wp`, which is mapped to the class `job` (cf. Section 5.2.2) in the task ontology. All twelve work-packages are then defined as a subclass of class `cipher-wp` and they are instantiated to represent their application-specific values.

The following box shows how the `work-package-1` is formalised in OCML. The representation of other work-packages can be realised along the same lines.

```

(def-class CIPHER-WP (job))

(def-class WORK-PACKAGE-1 (cipher-wp) ?wp-1
)

(def-class WORK-PACKAGE-1-ACTIVITY (activity) ?wp-1-activity
)

(def-class WORK-PACKAGE-1-TIME-RANGE (job-time-range) ?wp-1-time-range
)

(def-instance WP-1 work-package-1
  ((has-activities project-management-1)
   (requires-resource CO1-PERSON-1-MONTH-1 CO1-PERSON-1-MONTH-2
                     CO1-PERSON-1-MONTH-3 CO1-PERSON-1-MONTH-4
                     CO1-PERSON-1-MONTH-5 CO1-PERSON-1-MONTH-6
                     CO1-PERSON-1-MONTH-7 CO1-PERSON-1-MONTH-8
                     CO1-PERSON-1-MONTH-9 CO1-PERSON-1-MONTH-10
                     CO1-PERSON-1-MONTH-11 CO1-PERSON-1-MONTH-12
                     CO1-PERSON-1-MONTH-13 CO1-PERSON-1-MONTH-14
                     CO1-PERSON-1-MONTH-15 CR2-PERSON-1-MONTH-1
                     CR2-PERSON-1-MONTH-2 CR3-PERSON-1-MONTH-1
                     CR3-PERSON-1-MONTH-2 CR4-PERSON-1-MONTH-1
                     CR5-PERSON-1-MONTH-1 CR5-PERSON-1-MONTH-2
                     CR6-PERSON-1-MONTH-1)
   (has-time-range wp-1-time-range)
   (has-load 24)))

(def-instance PROJECT-MANAGEMENT-1 work-package-1-activity
  ((has-time-range project-management-1-time-range)
   (requires-resource CO1-PERSON-1-MONTH-1 CO1-PERSON-1-MONTH-2
                     CO1-PERSON-1-MONTH-3 CO1-PERSON-1-MONTH-4
                     CO1-PERSON-1-MONTH-5 CO1-PERSON-1-MONTH-6
                     CO1-PERSON-1-MONTH-7 CO1-PERSON-1-MONTH-8
                     CO1-PERSON-1-MONTH-9 CO1-PERSON-1-MONTH-10
                     CO1-PERSON-1-MONTH-11 CO1-PERSON-1-MONTH-12
                     CO1-PERSON-1-MONTH-13 CO1-PERSON-1-MONTH-14
                     CO1-PERSON-1-MONTH-15 CR2-PERSON-1-MONTH-1
                     CR2-PERSON-1-MONTH-2 CR3-PERSON-1-MONTH-1
                     CR3-PERSON-1-MONTH-2 CR4-PERSON-1-MONTH-1
                     CR5-PERSON-1-MONTH-1 CR5-PERSON-1-MONTH-2
                     CR6-PERSON-1-MONTH-1)
   (has-duration project-management-1-duration)
   (has-load 24)))

(def-instance PROJECT-MANAGEMENT-1-DURATION duration
  ((has-magnitude 29)
   (has-unit-of-measure month)))

(def-instance WP-1-TIME-RANGE work-package-1-time-range
  ((has-earliest-start-time (new-instance 'time-point '((year-of 2000)
                                                         (month-of 1))))
   (has-latest-end-time (new-instance 'time-point '((year-of 2002)
                                                         (month-of 6))))))

(def-class CO-ORDINATOR-1-PERSON-1 (resource))

(def-instance CO1-PERSON-1-MONTH-1 co-ordinator-1-person-1
  ((has-availability col-person-availability)
   (has-capacity 1)))

```

### 8.2.2 Modelling constraints and preference

The CIPHER application is formulated based on the following constraints and a preference.

- **End-time-compliance:** This constraint is common to all the twelve work-packages stating that each work-package must finish exactly on its end time and not earlier. A set of twelve end-time-compliance constraints are defined to impose this constraint. The following box shows the OCML definition of this constraint imposed on work-package-1.

```

(def-class CIPHER-JOB-CONSTRAINT (constraint))

(def-instance END-TIME-COMPLIANCE-WORK-PACKAGE-1 cipher-job-constraint
  ((applicable-to-jobs `(setofall ?x (work-package-1 ?x))
    (has-expression (kappa (?sc)
      (and (schedule-model ?sc)
        (has-time-range ?x ?wp-1-time-range)
        (has-latest-end-time ?wp-1-time-range ?let)
        (= (the-last-time-point-in-time-range
          ?wp-1-time-range) ?ltp)
        (time-points-equal ?let ?ltp)))))))

```

In order to check whether each work-package finishes on its end time, an application-specific function called `the-last-time-point-in-time-range` was defined. This function is used to retrieve the last time point from the time interval of each work-package and then the relation called `time-points-equal` is used to state that the last time point in a work-package interval must be equal to the latest end time of a work-package.

- **Coverage-constraint:** This constraint is also common to all the work-packages, stating that the 'idle time' ought to be minimised and every month of every work-package must be covered by at least one resource. The following two application-specific functions are defined to formalise this constraint: `all-time-points-in-interval` and `fetch-next-time-point`. The former function is used to retrieve all the time points from the work-package time interval such that it can be checked whether each time point is occupied by a resource. The latter function `fetch-next-time-point` is used in order to retrieve the next time point of a currently retrieved time point from the work-package interval, such that each next time point is parsed to `all-time-points-in-interval` to check its occupancy. The following box shows the OCML definitions of `coverage-constraint` and the two functions used to formalise this constraint.

```

(def-instance COVERAGE-CONSTRAINT cipher-job-constraint
  ((applicable-to-jobs '(setofall ?x (cipher-job ?x)))
   (has-expression (kappa (?sc)
                           (forall (?j ?a)
                             (=> (and (cipher-job ?j has-activities ?ca)
                                       (has-time-range ?ca ?jtr)
                                       (has-earliest-start-time ?jtr ?est)
                                       (has-latest-end-time ?jtr ?let)
                                       (month-in-time ?mit)
                                       (= (all-time-points-in-interval
                                           ?jtr ?mit) ?all))))
                                (forall (?tp)
                                  (=> (and (member ?tp ?all)
                                            (member (cipher-job ?ca ?r ?jtr2)
                                                    ?sc)
                                            (time-point-within-interval
                                             ?tp ?jtr))))
                                (forall (?all)
                                  (=> (not (= (the-slot-value
                                                  ?ca requires-resource) 0))))))))))

(def-function ALL-TIME-POINTS-IN-INTERVAL (?interval ?unit-of-measure)
  :constraint (and (job-time-range ?interval)
                  (unit-of-measure ?unit-of-measure))
  :body (and (has-earliest-start-time ?interval ?est1)
             (has-unit-of-measure ?est1 ?uom1)
             (has-latest-end-time ?interval ?let1)
             (has-unit-of-measure ?let1 ?uom2)
             (cons ?uom1 (FETCH-NEXT-TIME-POINT ?uom1 ?interval ?uom2 ?let1))))

(def-function FETCH-NEXT-TIME-POINT (?current-tp ?time-interval ?unit-of-measure
                                     ?last-tp)
  "This function retrieves the next time point of an existing time point."
  :constraint (and (unit-of-measure ?unit-of-measure)
                  (time-point ?current-tp has-unit-of-measure ?unit-of-measure)
                  (job-time-range ?time-interval)
                  (time-point ?last-tp has-unit-of-measure ?unit-of-measure))
  :body (in-environment
         ((?next-tp . (has-earliest-start-time ?time-interval ?current-tp)))
         (if
          (time-points-equal ?next-tp ?last-tp)
          ?last-tp
          (cons ?next-tp
                (fetch-next-time-point ?next-tp ?time-interval
                                       ?unit-of-measure ?last-tp))))))

```

- **Resource-availability-constraint:** This states that the correct numbers of project persons are required by each work-package for its successful completion. The project staff requirement of each work-package is imposed by complying with the data given in Table 8.3. A set of 73 constraints were defined to specify the staff requirements of all the work-packages. The following box shows the OCML definition of one such constraint imposed on work-package-1. The resource availability constraint for other work-packages can be realised analogously.

```

(def-instance CO1-RESOURCE-FOR-WP-1 cipher-job-constraint
  ((applicable-to-jobs '(setofall ?x (work-package-1 ?x)))
   (has-expression (kappa (?sc)
                           (exists ?r
                                (and
                                 (co1-resource ?r)
                                 (member (?work-package-1 ?r
                                         ?wp-1-activity ?wp-1-time-range) ?sc)
                                 (= (length ?r) 15)))))))

```

- **Competence-matching-preference:** Some people are better at certain tasks than others, so a schedule should satisfy this competence matching criterion.

Although, the CIPHER application represented quite a complex distribution of the project personnel over work-packages, our task ontology has provided an adequate leverage to capture this knowledge precisely. Moreover, the resource-capacity axiom (cf. Section 5.2.3.1) from the task ontology allowed us to maintain the unary capacity associated with every staff member. Different classes from the task ontology, such as job, activity, resource, duration, and job-time-range also provided the required level of detail and formalism to model the application-specific knowledge precisely. Nevertheless, a few new application-specific functions were defined to formalise the ‘end-time-compliance’ constraint and ‘coverage-constraint’. In the following section, we describe how a complete schedule for the CIPHER application was constructed.

### 8.2.3 Construction of a complete schedule by Propose & Backtrack

The primary goal of the CIPHER application was to construct a complete schedule quickly to see whether the project could be completed in a given period. As described in Section 7.4, because the Propose & Backtrack method can be used to construct a complete schedule, we applied this method to construct a schedule for CIPHER. This method was configured by defining the application-specific operators to assign resources to work-packages and by providing the application-specific knowledge to select a correct focus.

#### 8.2.3.1 Construction of the operators

Two types of operators, cipher-resource-operator and cipher-time-range-operator, were defined to assign work-packages to project staff and time ranges.

The cipher-resource-operator is defined in such a way that the staff requirements of each work-package are maintained throughout the schedule construction in accordance with Table 8.3. As described earlier one of the primary requirement of the CIPHER project is to assign the correct number of project-months for the timely completion of the work-packages. Therefore while defining cipher-resource-operator we also complied with resource-availability-constraint to make sure that the correct numbers of project members were available for executing the work-packages. Cipher-time-range-operator is defined in such a way that the work-packages can be finished exactly on their end times and not earlier. The following box shows the OCML definitions of the operators defined for work-package-1.

```

(def-class CIPHER-RESOURCE-OPERATOR
  (multiple-schedule-extension-resource-operator))

(def-class CIPHER-TIME-RANGE-OPERATOR
  (multiple-schedule-extension-time-range-operator))

(def-instance COL-RESOURCE-FOR-WORK-PACKAGE-1 cipher-resource-operator
  ((applicable-to-jobs '(setofall ?x (work-package-1-job ?x)))
   (has-body (lambda (?x ?sc)
                (the ?col-resource
                  (and (col-resource ?col-resource)
                       (= (length ?col-resource) 15)))))))

(def-instance WORK-PACKAGE-1-TIME-RANGE-OPERATOR cipher-time-range-operator
  ((applicable-to-jobs '(setofall ?x (work-package-1-job ?x)))
   (has-body (lambda (?x ?sc)
                (the ?wp-1-time-range
                  (work-package-1-time-range ?wp-1-time-range))))))

(tell (SCHEDULE-OPERATOR-ORDER col-resource-for-work-package-1
                                work-package-1-time-range-operator))

```

Finally, the relation `schedule-operator-order` (cf. Section 6.2.2) is instantiated to determine the order in which different operators are applied to assign each work-package.

### 8.2.3.2 Focus and operator selection, and schedule generation

Because the CIPHER application required all its work-packages to finish exactly on their end time, while selecting a correct focus we complied with this application-specific knowledge. The job selection method `job-selection-based-on-latest-end-time` (cf. Section 6.3.5) is used in order to select a focus based on this knowledge. According to this method first all the work-packages are sorted according to their latest end time and then the first work-package from the sorted list is selected as a focus in each cycle.

Having completed the configuration of Propose & Backtrack, we ran the application until all the work-packages were assigned to the required number of person-months and time ranges. Solving this application by using Propose & Backtrack turned out to be very efficient because the solution space was very dense. Therefore, very little search was required to reach a solution state. A complete schedule for the CIPHER application was constructed by generating 1342 schedule states. According to our focus selection strategy all the work-packages were instantiated according to their latest end time, which helped to complete all the work-packages successfully within their latest end time. As a result, the application of our method satisfied one of the important solution criteria of the CIPHER application. Also once the work-packages were sorted based on their latest end time then they were selected almost linearly, and therefore, a solution state was reached without any backtracking which resulted in 100% efficiency. More importantly, no other constraints were violated while constructing a schedule. Therefore, a complete and consistent solution was returned once all the work-packages were assigned.

8.3 The daily ship-maintenance application

8.3.1 Construction of a task model

The daily ship-maintenance application is a real world problem consisting of thirteen ship-maintenance jobs, which have to be assigned to thirteen ship-maintenance resources. The ship-maintenance resources are categorised into two groups: maintenance personnel and maintenance machinery. A schedule for this application is constructed on a daily basis and the working hours of each day are from 9:00am to 18:00pm.

Each ship-maintenance job has a number of ship-maintenance activities associated with it, which need to be accomplished to complete the job. All the activities must be completed within the fixed duration of the job. Each ship-maintenance job also has a specific requirement for the ship-maintenance resources on which it must be assigned for its completion. Finally the ship-maintenance jobs require a specific *number* of ship-maintenance resources to complete the maintenance activities. Table 8.3 represents a data used to formalise the ship-maintenance jobs. The column labelled ‘load’ in Table 8.4 represents the number of ship-maintenance resources required by each ship-maintenance job for its completion.

Table 8.4. Data used to formalise the ship-maintenance jobs.

Ship-maintenance jobs	Description of the activities	Resource requirement	Duration (Minutes)	Load
C4B9UQN	Inspect-citric-acid Inspect-urinal-to-ensure-citric-acid	AN/FN/SN	48	2
A4C2DCN	Inspect-balance-pressure-proportional Inspect-gauge-level Inspect-the-level-of-cooler-in-degasser	AN/FN/SN	60	1
A4GDBHN	Test-mode-control	FC2	24	1
A4GDBHN-1-1	Test-LCP-keyboard-entry	FC2-1	10	1
A4GDBHN-2-1	Test-ship-heading-readout	FC2-2	10	1
A4GDBHN-3	Test-audible-alarm-and-mount-safety	FC2-3	10	1
A4GDBHN-4	Perform-system-	FC2-4	10	1

	operability-test			
26N45CN	Clean-galley-or-pantry-vent	FN/SN	40	1
C5DVYGN	Inspect-all-periferral-ammunition-equipments	GMG3	36	1
51GEPVN	Perform-daily-ability-test	GMG3-2	48	1
B24FDXN	Inspect-water-level-in-bilge	MM/EN3	35	1
B24FDXN-1	Inspect-oil-level-in-upper-gravity-tan	MM/EN3	24	1
40KL83N	Conduct-lamp-and-alarm-test	RMSN	48	1

Each ship-maintenance resource has a special competence, which indicates the specific types of ship-maintenance jobs it can handle. The ship-maintenance resources also have a fixed availability period and the ship-maintenance jobs must be executed only within this period. Finally, all the ship-maintenance resources have a fixed capacity, which determines the total number of ship-maintenance jobs each ship-maintenance resource can handle at any one time. Table 8.5 represents the maximum capacity of each ship-maintenance resource.

Table 8.5. Maximum capacity of the ship-maintenance resources.

Daily ship-maintenance resource	Capacity
AN/FN/SN	2
DC/HT2	2
FC2	1
FC2-1	1
FC2-2	1
FC2-3	1
FC2-4	1
FN/SN	2
GMG3	3
GMG3-2	3
MM/EN3	1
MM/EN3-2	1
RMSN	2

As an example the following box shows the OCML definitions representing how the ship-maintenance job called C4B9UQN and its attributes are represented in the task model.

```
(def-class DAILY-SHIP-MAINTENANCE-JOB (job))

(def-class DAILY-SHIP-MAINTENANCE-RESOURCE (resource))

(def-class DAILY-SHIP-MAINTENANCE-ACTIVITY (activity))

(def-class SHIP-MAINTENANCE-JOB-TIME-RANGE (job-time-range))

(def-class C4B9UQN-JOB (daily-ship-maintenance-job))

(def-class C4B9UQN-ACTIVITY (daily-ship-maintenance-activity))

(def-class C4B9UQN-JOB-TIME-RANGE (ship-maintenance-job-time-range))

(def-instance C4B9UQN C4B9UQN-JOB
  ((has-activities inspect-citric-acid inspect-urinal-to-ensure-citric-acid)
   (requires-resource AN/FN/SN)
   (has-time-range C4B9UQN-time-range)
   (has-load 2)))

(def-instance INSPECT-URINAL-TO-ENSURE-CITRIC-ACID C4B9UQN-activity
  ((has-duration inspect-urinal-to-ensure-citric-acid-duration)
   (requires-resource AN/FN/SN)
   (has-time-range inspect-urinal-to-ensure-citric-acid-time-range)))

(def-instance C4B9UQN-TIME-RANGE C4B9UQN-job-time-range
  ((has-latest-start-time (new-instance 'time-point '(hour-of 09)
                                          (minute-of 00))))
   (has-latest-end-time (new-instance 'time-point '((hour-of 09)
                                                    (minute-of 48))))))

(def-instance INSPECT-CITRIC-ACID duration
  ((has-magnitude 18)
   (has-unit-of-measure minute)))

(def-instance INSPECT-URINAL-TO-ENSURE-CITRIC-ACID-DURATION duration
  ((has-magnitude 30)
   (has-unit-of-measure minute)))
```

### 8.3.1.1 Modelling constraints and requirements

The following constraints and requirements were elicited in the context of the ship-maintenance application.

- **Job precedence constraint:** This constraint is common to all the ship-maintenance jobs, and imposes a strict precedence ordering among all the jobs. A set of thirteen job precedence constraints have been defined to apply this constraint to all the ship-maintenance jobs. In order to impose the precedence ordering among any two ship-maintenance jobs the relation called job-precedes (cf. Section 5.2.2.3) is used from the task ontology, which states that if the latest end time of ship-maintenance job, say  $S_1$  is before the earliest start time of ship-maintenance job, say  $S_2$ , then  $S_1$  precedes  $S_2$ . The following box shows the OCML definition of the job precedence constraint imposed between ship-maintenance jobs C4B9UQN and A4C2DCN.

```

(def-class DAILY-SHIP-CONSTRAINT (constraint))

(def-instance PRECEDENCE-AMONG-C4B9UQN-A4C2DCN daily-ship-constraint
  ((applicable-to-jobs '(C4B9UQN A4C2DCN))
   (has-expression (kappa (?sc)
                           (exists ?C4B9UQN-job
                                     (and (C4B9UQN-job ?C4B9UQN-job)
                                           (has-time-range
                                            ?C4B9UQN-job ?C4B9UQN-job-time-range)
                                           (= the ?let1 (has-latest-end-time
                                                            ?C4B9UQN-job-time-range
                                                            ?let1))
                                           (exists ?A4C2DCN-job
                                                     (and (A4C2DCN-job ?A4C2DCN-job)
                                                           (has-time-range
                                                            ?A4C2DCN-job
                                                            ?A4C2DCN-job-time-range)
                                                           (= the ?let2 (has-latest-end-
time ?A4C2DCN-job-time-range ?let2))
                                                           (if (precedes ?let1 ?let2)
                                                             (job-precedes
                                                              ?C4B9UQN-job
                                                              ?A4C2DCN-job)))))))))))

```

- **Daily frequency of ship maintenance jobs:** This constraint is common to all the thirteen ship-maintenance jobs, which states that all the ship-maintenance jobs must finish within their daily working hours, i.e. between 9:00am to 18:00pm. The following box shows the OCML definition of one such constraint imposed on a ship-maintenance job called, 40KL83N.

```

(def-instance DAILY-FREQUENCY-OF-40KL83N-job daily-ship-constraint
  ((applicable-to-jobs '(setofall ?x (40KL83N-job ?x)))
   (has-expression (kappa (?sc)
                           (exists ?40KL83N-job
                                     (and (40KL83N-job ?40KL83N-job)
                                           (has-time-range ?40KL83N-job ?40KL83Njtr)
                                           (daily-schedule-horizon ?daily-sc-horizon)
                                           (time-ranges-not-exceed
                                            ?40KL83Njtr ?daily-sc-horizon)))))))

```

- **Job priority requirement:** This requirement states that if more than one ship-maintenance job is consuming the same ship-maintenance resource then the ship-maintenance job with the higher number of activities needs to be given priority for its execution. In order to specify the job priority requirement, we used the function called `number-of-activities-within-job` (cf. Appendix 1), which retrieved all the ship-maintenance activities associated with each ship-maintenance job, and then the `higher-priority-job-based-on-activities` relation (cf. Section 5.2.2.3) is used to state a condition according to which the ship-maintenance job with the higher number of activities is given priority. The following box shows the OCML definition of one such requirement associated with the ship-maintenance jobs C4B9UQN and A4C2DCN. The job priority requirement for the other two ship-maintenance jobs can be realised analogously.

```
(def-class DAILY-SHIP-REQUIREMENT (requirement))

(def-instance JOB-PRIORITY-AMONG-C4B9UQN-AND-A4C2DCN daily-ship-requirement
  ((applicable-to-jobs '(C4B9UQN A4C2DCN))
   (has-expression (kappa (?sc)
    (= (number-of-activities-within-job
      ?C4B9UQN-job) ?C4B9UQN-activity)
      (= (number-of-activities-within-job
        ?A4C2DCN-job) ?A4C2DCN-activity)
      (if (> (length ?C4B9UQN-activity)
              (length ?A4C2DCN-activity))
          (higher-priority-job-based-on-activities
            ?C4B9UQN-job ?A4C2DCN-job)))))))
```

This concludes our discussion about how the task model for the daily ship-maintenance application is constructed. In the following section, we describe how we constructed a schedule for the application.

### 8.3.2 Construction of a schedule by using Generic-Schedule

In order to construct a complete schedule for this application the Generic-Schedule method from our library was applied.

#### 8.3.2.1 Operator construction for the daily-ship schedule

The Generic-Schedule method was configured by defining the following two types of application-specific operators: daily-resource-operator and daily-time-range-operator. The daily-resource-operator is used to assign ship-maintenance jobs to their respective ship-maintenance resources. This operator is constructed by complying with the ship-maintenance resource requirement of each ship-maintenance job as given in Table 8.3, and also by maintaining the total number of resources required by each ship-maintenance job as given by the column ‘load’ in Table 8.3. The daily-time-range-operator is defined in such a way that a correct time range can be assigned to the ship-maintenance jobs. The following box shows the OCML definitions of the operators defined for assigning the ship-maintenance job called C4B9UQN.

```
(def-instance C4B9UQN-to-AN/FN/SN-RESOURCE daily-resource-operator
  ((applicable-to-jobs '(setofall ?x (C4B9UQN-job ?x)))
   (has-body (lambda (?x ?s)
    (the ?AN/FN/SN-resource
      (and (AN/FN/SN-resource ?AN/FN/SN-resource)
            (= (number-of-resources-for-job ?C4B9UQN-job) 1)))))))

(def-instance C4B9UQN-to-C4B9UQN-TIME-RANGE daily-time-range-operator
  ((applicable-to-jobs '(setofall ?x (C4B9UQN-job ?x)))
   (has-body (lambda (?x ?s)
    (the ?C4B9UQN-job-time-range
      (C4B9UQN-job-time-range ?C4B9UQN-job-time-range))))))

(tell (schedule-operator-order C4B9UQN-to-AN/FN/SN-resource
  C4B9UQN-to-C4B9UQN-time-range))
```

### 8.3.3 Focus and operator selection

This application requires all the ship-maintenance jobs to be completed within their daily frequency. To comply with this requirement we use the job selection method called job-

selection-based-on-start-time (cf. Section 6.3.5), which first sorts all the ship-maintenance jobs on the basis of their earliest start time, and then selects the first job. Once the focus is selected then all the operators necessary to assign the selected focus are collected and then sorted by instantiating the relation `schedule-operator-order` (cf. Section 6.2.2).

A complete schedule for this application was constructed by generating 852 schedule states. While constructing a complete schedule the search backtracked five time because the 'job-priority-requirement' imposed on the ship-maintenance jobs: C4B9UQN and A4C2DCN was violated. Therefore, the overall efficiency of schedule construction by using `Generic-Schedule` was 75%. The efficiency is calculated as the ratio between the size of the minimal search space required to solve the application to that of the effective search space navigated to reach a solution. The main reason why job-priority-requirement was violated was because, according to our focus selection strategy, a ship-maintenance job with the earliest start time gets assigned first. However this focus selection strategy conflicted with job-priority-requirement, which required the ship-maintenance job with highest number of activities to be assigned first. It means that A4C2DCN-job should have been assigned before assigning C4B9UQN-job because the former job has three activities associated with it while only two activities are associated with the latter job. The following box shows the synoptic trace of the behaviour of `Generic-Schedule` while constructing a complete schedule for this application. The box below also shows how the complete schedule looks like once all the daily ship-maintenance jobs are assigned.

```

----- Enter task EVALUATE-SCHEDULE-STATE855 with arguments
(HAS-SCHEDULE-STATE SCHEDULE-STATE852)

----- Enter task EVALUATE-HARD-CONSISTENCY856 with
arguments (HAS-SCHEDULE-STATE SCHEDULE-STATE852)

----- Exit task EVALUATE-HARD-CONSISTENCY856 -> NIL

----- Enter task EVALUATE-COMPLETENESS857 with arguments
(HAS-SCHEDULE-STATE SCHEDULE-STATE852)

----- Exit task EVALUATE-COMPLETENESS857 -> (STATE-COMplete
SCHEDULE-STATE852)

OCML 18 : 1 > (describe-instance 'schedule-state852)

Instance SCHEDULE-STATE852 of class SCHEDULE-STATE

HAS-SCHEDULE-MODEL: ((A4GDBHN FC2 TEST-MODE-CONTROL A4GDBHN-1-TIME-RANGE) (A4GDBHN-1-1
FC2-1 TEST-LCP-KEYBOARD-ENTRY A4GDBHN-1-1-TIME-RANGE) (A4GDBHN-2-1 FC2-2 TEST-SHIP-
HEADING-READOUT A4GDBHN-2-1-TIME-RANGE) (A4GDBHN-3 FC2-3 TEST-AUDIBLE-ALARM-AND-MOUNT-
SAFETY A4GDBHN-3-TIME-RANGE) (A4GDBHN-4 FC2-4 PERFORM-SYSTEM-OPERABILITY-TEST A4GDBHN-4-
TIME-RANGE) (26N45CN FN/SN CLEAN-GALLEY-OR-PANTRY-VENT 26N45CN-TIME-RANGE) (C5DVYGN GMG3
INSPECT-ALL-PERIFERRAL-AMMUNITION-EQUIPMENTS C5DVYGN-TIME-RANGE) (51GEPVN GMG3-2 PERFORM-
DAILY-ABILITY-TEST 51GEPVN-TIME-RANGE) (B24FDXN MM/EN3 INSPECT-WATER-LEVEL-IN-BILGE
B24FDXN-TIME-RANGE) (B24FDXN-1 MM/EN3 INSPECT-OIL-LEVEL-IN-UPPER-GRAVITY-TANK B24FDXN-1-
TIME-RANGE) (40KL83N RMSN CONDUCT-LAMP-AND-ALARM-TEST 40KL83N-TIME-RANGE) (A4C2DCN
AN/FN/SN INSPECT-BALANCE-PRESSURE-PROPORTIONER A4C2DCN-TIME-RANGE) (C4B9UQN AN/FN/SN
INSPECT-URINAL-TO-ENSURE-CITRIC-ACID C4B9UQN-TIME-RANGE))

```

In order to fix the requirement violation that occurred while constructing a complete schedule we applied the P&Rf method from the library.

### 8.3.4 Fixing the requirement violation by using the P&Rf method

The propose phase of the P&Rf method is configured straightforwardly from Generic-Schedule. The only difference between the configuration process of Generic-Schedule and the propose phase is that a job dependency network is constructed explicitly to represent the dependencies between assignments. The job dependency network is modelled by using the relations `job-depends-on` and `job-affects` (cf. Section 6.2.3), which are part of the method ontology.

#### 8.3.4.1 Construction of the feasibility-restoration operator

In order to fix the requirement violation that occurred while constructing a schedule a new type of application-specific operator called `ship-restoration-operator` was defined while configuring the `restore-feasibility` phase of the method. This operator is constructed in such a way that the requirement violations which occurred while constructing a schedule can be fixed by changing the assignment strategy of the jobs involved in the conflict. By using `ship-restoration-operator` the priority of the culprit jobs can be changed by ranking them on the basis of the highest number of activities. The function called `number-of-activities-within-job` is used to retrieve all the activities associated with the `ship-maintenance` jobs, and then the relation called `higher-priority-job-based-on-activities` (cf. Section 5.2.2.3) is used to determine the priority of the jobs based on the number of activities. The following box shows the OCML definitions of `feasibility-restoration-operator` defined to fix the requirement violation, which occurred between jobs `C4B9UQN` and `A4C2DCN`.

```
(def-class SHIP-RESTORATION-OPERATOR (feasibility-restoration-operator))

(def-instance JOB-PRIORITY-BASED-ON-ACTIVITIES ship-restoration-operator
  ((applicable-to-requirements '(JOB-PRIORITY-AMONG-C4B9UQN-and-A4C2DCN))
   (applicable-to-jobs '(C4B9UQN A4C2DCN))
   (has-body '(lambda (?daily-ship-maintenance-job ?sc)
                 (the ?A4C2DCN-job
                     (exists ?A4C2DCN-job
                          (and (A4C2DCN-job ?A4C2DCN-job)
                               (= (number-of-activities-within-job
                                   ?A4C2DCN-job) ?l1)
                               (exists ?C4B9UQN-job
                                    (and (C4B9UQN-job ?C4B9UQN-job)
                                         (= (number-of-activities-within-
                                             ?C4B9UQN-job) ?l2)
                                         (if (> ?l1 ?l2)
                                             (higher-priority-job-based-on-
                                                ?A4C2DCN-job ?C4B9UQN-
                                                job))))))))))))))
```

### 8.3.4.2 Construction of a feasible schedule

Once the configuration of the P&Rf method was completed then we ran this application to construct a schedule. Once a complete schedule is constructed by executing the propose phase then in the restore-feasibility phase the violated requirement job-priority-among-C4B9UQN-and-A4C2DCN is selected as a candidate focus. Having selected a focus, then the operator called job-priority-based-on-activities is selected to fix this violated requirement.

It was observed that the new operator job-priority-based-on-activities successfully fixed the requirement violation occurred between the ship-maintenance jobs C4B9UQN and A4C2DCN. In the new solution generated by applying job-priority-based-on-activities, the ship-maintenance job A4C2DCN was assigned before the ship-maintenance job C4B9UQN. It was observed that a solution constructed by using the P&Rf method was more robust as compared to the one constructed by Generic-Schedule, because no other part of a complete schedule was affected while fixing the existing requirement violation. Moreover, other constraints and requirements imposed on the daily ship-maintenance application were also maintained throughout the schedule construction. As a result, a complete and feasible solution schedule for this application was constructed by generating 949 schedule states.

## 8.4 The weekly ship-maintenance application

Here we describe the validation of the library on the weekly ship-maintenance application, which is again a real-life scheduling application. As in the case of the daily ship-maintenance application, a schedule for the weekly ship-maintenance application is also constructed to perform different types of ship-maintenance activities. However, the weekly ship-maintenance application is more complex in nature compared to the daily ship-maintenance application due to the more complex nature of the relevant constraints and requirements.

### 8.4.1 Construction of a task model

The weekly ship-maintenance application can be described as an assignment of the ship-maintenance jobs to the ship-maintenance resources within specific time ranges such that a *complete* and a *consistent* schedule is constructed. The working hours for each day are from 9:00am to 18:00pm.

This application consists of twenty one ship-maintenance jobs, which have to be assigned on nineteen ship-maintenance resources. The ship-maintenance jobs have a specific requirement for the ship-maintenance resources and they can be assigned only on

these ship-maintenance resources for their execution. All the ship-maintenance jobs also have a number of activities associated with it, which must be executed to accomplish the ship-maintenance jobs. Each ship-maintenance resource has a specific competence, which determines the specific types of ship-maintenance jobs it can handle for their execution. The ship-maintenance resources also have a fixed capacity, which determines the total number of ship-maintenance jobs they can handle at any one time. Finally, each ship-maintenance resource is available only during a restricted period. Table 8.6 shows the data used to formalise the weekly ship-maintenance jobs.

Table 8.6. Data used to formalise the ship-maintenance jobs.

Ship-maintenance jobs	Description of the activities	Resource requirement	Activity Duration (Minutes)	Load	Resource capacity
12B3HTN	Turn-pump-shaft-by-hand	ABF/BT/EN2-resource	48	1	2
63A2TFN	Strip-JP5-service-tank	ABF/EN/GSM/M2-resource	144	1	3
36A2RDN	Inspect-seal-tank-water-level	ABF/EN/MM3-resource	48	1	3
44B9URN	Test-operate-and-inspect-flushometer	AN/FN/SN-resource	48	1	1
C23HZAN	Inspect-gearcase-oil-level-idle-winch	BM3-resource	240	1	2
359BZUN	Flush-hellan-seawater-strainer	BT/EN/GSM/M3-resource	48	1	1
47K78GM	Lubricate-pump	BT/EN/MM3-resource	96	1	4
17A8UBN	Inspect-bubbler-liquid-level	BT/MM3-resource	48	1	5
64M44EN	Test-clean-and-inspect-flame-scanner	BT2-resource	240	1	3
B2B7TCN	Visually-inspect-pump-unit	DC/HT3-resource	100	1	2

B2A7SAN	Visually-inspect-pump-unit-2	DC/HT3-1-resource	100	1	1
A4C2EHN	Test-and-operate-AFFF-concrete-pump-assembly	DC/HT3-2-resource	100	1	3
34A1JWN	Inspect-differential-pressure/pressure-drop	DCPO-resource	90	1	2
36A1JVN	Inspect-and-test-relay-operated-lantern	DCPO-1-resource	90	1	1
36W29WN	Inventory-and-inspect-fire-hose-station-eqmt	DCPO-2-resource	90	1	1
36W29XN	Accomplish-functional-test-of-portable-lantern	DCPO-3-resource	90	1	2
36W31CN	Inspect-and-test-relay-operated-lantern2	DCPO-4-resource	90	1	2
482YTTN	Test-signal-and-navigation-lights	EM3-resource	144	1	3
628URAN	Test-running-light-telltale-panel	EM3-1-resource	144	1	1
51A1BNN	Inspect-microwave-oven	DCPO-4-resource	96	1	2
266TFEN	Accomplish-functional-test-of-engine-battery	EM3-3-resource	48	1	3

### 8.4.1.1 Modelling ship-maintenance jobs and resources

The following box shows how the weekly ship-maintenance job 12B3HTN is represented in the application. The formalisation of the other weekly ship-maintenance jobs can be realised analogously.

```
(def-class WEEKLY-SHIP-MAINTENANCE-JOB (job))

(def-class WEEKLY-SHIP-MAINTENANCE-ACTIVITY (activity))

(def-class SHIP-MAINTENANCE-JOB-TIME-RANGE (job-time-range))

(def-class 12B3HTN-JOB (weekly-ship-maintenance-job))

(def-class 12B3HTN-ACTIVITY (weekly-ship-maintenance-activity))

(def-class 12B3HTN-JOB-TIME-RANGE (ship-maintenance-job-time-range))

(def-instance 12B3HTN 12B3HTN-job
  ((has-activities turn-pump-shaft-by-hand)
   (requires-resource ABF-BT-EN2)
   (has-time-range 12B3HTN-time-range)
   (has-load 1)))

(def-instance turn-pump-shaft-by-hand 12B3HTN-activity
  ((has-duration turn-pump-shaft-by-hand-duration)
   (requires-resource ABF-BT-EN2)
   (has-time-range turn-pump-shaft-by-hand-time-range)))

(def-instance 12B3HTN-time-range 12B3HTN-job-time-range
  ((has-latest-start-time (new-instance 'time-point '((hour-of 09)
                                                         (minute-of 10))))
   (has-latest-end-time (new-instance 'time-point '((hour-of 9)
                                                         (minute-of 58))))))
```

As in the case of the weekly ship-maintenance jobs, an application-specific class called `weekly-ship-maintenance-resource` is defined to represent the weekly ship-maintenance resources. The following box shows how `ABF/BT/EN2-resource` is formalised in the application.

```
(def-class WEEKLY-SHIP-MAINTENANCE-RESOURCE (resource))

(def-class ABF-BT-EN2-RESOURCE (weekly-ship-maintenance-resource))

(def-instance ABF-BT-EN2 ABF-BT-EN2-resource
  ((has-job-belonging 12B3HTN)
   (has-availability ABF-BT-EN2-availability)
   (has-capacity 1)))

(def-instance ABF-BT-EN2-AVAILABILITY time-range
  ((has-start-time (new-instance 'time-point '((hour-of 09)
                                                         (minute-of 00))))
   (has-end-time (new-instance 'time-point '((hour-of 17)
                                                         (minute-of 00))))))
```

### 8.4.1.2 Modelling the constraints and requirements

The weekly ship-maintenance application includes the following constraints and requirements:

- **Resource capacity constraint:** This constraint is common to all the ship-maintenance resources. It states that each ship-maintenance resource has a fixed capacity as described in Table 8.5, which determines the total number of ship-maintenance jobs each ship-maintenance resource can handle. A set of nineteen resource capacity

constraints are defined to impose resource-capacity-constraint on all the nineteen ship-maintenance resources. To formalise these constraints, we used the function called `maximum-capacity-of-resource` (cf. Section 5.2.3), which retrieved all the weekly ship-maintenance jobs associated with a resource, and then an equality condition is imposed to limit the total number of weekly ship-maintenance jobs each ship-maintenance resource can handle according to the data given in Table 8.5. The following box shows the OCML definition of one such constraint imposed on the `ABF/BT/EN2-resource`.

```
(def-instance ABF-BT-EN2-RESOURCE-CAPACITY-CONSTRAINT weekly-ship-constraint
  ((applicable-to-resources '(setofall ?x (ABF-BT-EN2-resource ?x)))
   (has-expression (kappa (?sc)
                          (exists ?ABF-BT-EN2-resource
                                (and (ABF-BT-EN2-resource ?ABF-BT-EN2-resource)
                                     (member (?weekly-ship-maintenance-job
                                              ?weekly-ship-maintenance-activity
                                              ?ABF-BT-EN2-resource
                                              ?ship-maintenance-job-time-range)
                                              ?sc)
                                     (= (the ?c1 (maximum-capacity-of-resource
                                              ?ABF-BT-EN2-resource))
                                         2))))))))))
```

- **Daily frequency of ship maintenance jobs:** This constraint is common to all the twenty one ship-maintenance jobs, and states that all the ship-maintenance jobs must finish exactly within their daily working hours. In order to check whether all the ship-maintenance jobs comply with their daily time frequency, we used the relation `time-points-equal` from the Simple Time ontology, which states that the latest end time of each weekly ship-maintenance job must be equal to the end time of the daily working hour of a schedule. The following box shows the OCML definition of this constraint imposed on the `266TFEN` ship-maintenance job. The daily frequency constraint for other weekly ship-maintenance jobs can be realised analogously.

```
(def-instance DAILY-FREQUENCY-OF-266TFEN-JOB weekly-ship-constraint
  ((applicable-to-jobs '(setofall ?x (266TFEN-job ?x)))
   (has-expression (kappa (?sc)
                          (exists ?266TFEN-job
                                (and (266TFEN-job ?266TFEN-job)
                                     (has-time-range
                                      ?266TFEN-job ?266TFEN-jtr)
                                     (= ?the ?let (has-latest-end-time
                                              ?266TFEN-jtr ?let))
                                      (daily-time-range ?daily-time-range)
                                     (= the ?et (has-end-time
                                              ?daily-time-range ?et))
                                      (time-points-equal ?let ?et))))))))))
```

- **Job working hour constraint:** This constraint is again common to all the ship-maintenance jobs, and states that in the worse case scenario a ship-maintenance job may exceed its duration, by not more than 10 minutes as long as this does not violate the daily frequency of a schedule. The following box shows the OCML definition of this constraint.

```

(def-instance JOB-WORKING-HOUR weekly-ship-constraint
  ((applicable-to-jobs '(setofall ?x (weekly-ship-maintenance-job ?x)))
   (has-expression (kappa (?sc)
                           (exists ?x
                                     (and (weekly-ship-maintenance-job ?x)
                                           (has-time-range ?x ?xtr)
                                           (= (exceeded-duration-of-job ?xtr) ?dur-e)
                                           (duration-is-less-than-or-equal
                                            ?dur-e (10 minute))
                                           (exists ?dftr
                                                     (and (daily-working-hours ?dftr)
                                                           (= (time-range-duration
                                                            ?dftr) ?dur-dftr)
                                                           (not (duration-is-less-than-
or-equal ?dur-e ?dur-dftr)))))))))))

```

- **Job priority requirement:** This requirement is also common to all the weekly ship-maintenance jobs, and states that if any two ship-maintenance jobs share a same ship-maintenance resource for their execution, then a weekly ship-maintenance job with higher duration gets priority. To formalise this requirement, we used the function `job-time-range-duration`, which retrieved the duration of the ship-maintenance jobs and then the retrieved durations of the jobs are compared by using the relation `job-with-higher-priority` to determine their priority. The following box shows the OCML definition of this requirement.

```

(def-instance JOB-PRIORITY-BASED-ON-HIGHER-DURATION weekly-ship-requirement
  ((applicable-to-jobs '(setofall ?x (weekly-ship-maintenance-job ?x)))
   (has-expression (kappa (?sc)
                           (exists ?wm-job
                                     (and (member ?wm-job ?x)
                                           (member (?wm-job ?a1 ?r1 ?jtr1) ?sc)
                                           (requires-resource ?wm-job ?r1)
                                           (= (job-time-range-duration
                                            ?wm-job ?jtr1) ?d1)
                                           (exists
                                            ?wm-job2
                                            (and (member ?wm-job2 ?x)
                                                  (member (?wm-job2 ?a2 ?r1 ?jtr2)
                                                           ?sc)
                                                  (requires-resource ?wm-job2 ?r1)
                                                  (= (job-time-range-duration
                                                    ?wm-job2 ?jtr2) ?d2)
                                                  (if (> ?d1 ?d2)
                                                      (job-with-higher-priority
                                                       ?wm-job ?wm-job2))))))))))

```

This concludes our description of the task model of the weekly ship-maintenance application. In the following section, we will describe how a solution schedule for this application was constructed.

#### 8.4.2 Applying the Propose & Backtrack method

As described earlier one of the main goals of this application is to construct a complete schedule and therefore we first applied the Propose & Backtrack method from our library.

##### 8.4.2.1 Construction of the operators

The Propose & Backtrack method was configured by defining two types of application-specific operators - `weekly-ship-resource-operator` and `weekly-ship-time-range-operator`. The former operator is defined in order to assign weekly ship-

maintenance jobs to weekly ship-maintenance resources by maintaining the resource requirement of all the weekly ship-maintenance jobs as described in Table 8.5. Weekly-ship-resource-operator is defined in such a way that a correct time range can be assigned to all the weekly ship-maintenance jobs. The following box shows the OCML definitions of weekly-ship-resource-operator and weekly-ship-time-range-operator defined for the 482YTTN-job. The operators for the other weekly ship-maintenance jobs can be realised along the same lines.

```
(def-class WEEKLY-SHIP-RESOURCE-OPERATOR
  (multiple-schedule-extension-resource-operator))

(def-class WEEKLY-SHIP-TIME-RANGE-OPERATOR
  (multiple-schedule-extension-time-range-operator))

(def-instance 482YTTN-job-to-EM3-resource weekly-ship-resource-operator
  ((applicable-to-jobs '(setofall ?x (482YTTN-job ?x)))
   (has-body (lambda (?x ?s)
                (the ?EM3-resource
                    (EM3-resource ?EM3-resource))))))

(def-instance 482YTTN-job-482YTTN-time-range weekly-ship-time-range-operator
  ((applicable-to-jobs '(setofall ?x (482YTTN-job ?x)))
   (has-body (lambda (?x ?s)
                (the ?482YTTN-job-time-range
                    (482YTTN-job-time-range ?482YTTN-job-time-range))))))

(tell (schedule-operator-order 482YTTN-job-to-EM3-resource
                               482YTTN-job-482YTTN-time-range))
```

Finally, the relation `schedule-operator-order` (cf. Section 6.2.2) is instantiated to determine the order in which different operators are applied to assign a weekly ship-maintenance job.

### 8.4.3 The focus and operator selection knowledge

The focus selection task in this application is carried out by using the method `job-selection-based-on-least-number-of-activities` (cf. Section 6.3.5). In accordance with this method in each cycle a weekly ship-maintenance job with the least number of activities is selected as a focus. Once a correct focus is selected then all the operators are collected and sorted by instantiating the relation `schedule-operator-order` (cf. Section 6.2.2).

Once the configuration of the Propose & Backtrack is completed by determining how the focus can be selected, then we ran the weekly ship-maintenance application to construct its complete schedule. The following box shows the synoptic trace of the behaviour of the application. As it can be seen in the following box the complete schedule for the weekly ship-maintenance application was constructed by generating 1245 schedule states. We achieved 65% efficiency while constructing a schedule for this application. It was observed that while constructing a solution the search backtracked six times because the ‘daily frequency of ship-maintenance job constraint’ imposed on the following four weekly ship-

maintenance jobs - 482YTTN, 628URAN, 51A1BNN, and 266TFEN was violated as they violated their latest end time. In other words, the schedule was a complete but was not a consistent.

```

----- Enter task EVALUATE-COMPLETENESS1250
with arguments (HAS-SCHEDULE-STATE SCHEDULE-STATE1245)

----- Exit task EVALUATE-COMPLETENESS1250 -
> (STATE-COMplete SCHEDULE-STATE1245)

----- Enter task EVALUATE-FEASIBILITY1251
with arguments (HAS-SCHEDULE-STATE SCHEDULE-STATE1245)

----- Exit task EVALUATE-FEASIBILITY1251 ->
NIL

----- Exit task EVALUATE-SCHEDULE-STATE1248
-> NIL

OCML 27 : 1 > (describe-instance 'schedule-state1245)

Instance SCHEDULE-STATE1245 of class SCHEDULE-STATE

HAS-SCHEDULE-MODEL: ((12B3HTN ABF/BT/EN2 TURN-PUMP-SHAFT-BY-HAND 12B3HTN-TIME-RANGE)
(63A2TFN ABF/EN/GSM/MM2 STRIP-JP5-SERVICE-TANK 63A2TFN-TIME-RANGE) (36A2RDN ABF/EN/MM3
INSPECT-SEAL-TANK-WATER-LEVEL 36A2RDN-TIME-RANGE) (44B9URN AN/FN/SN TEST-OPERATE-AND-
INSPECT-FLUSHOMETER 44B9URN-TIME-RANGE) (C23HZAN BM3 INSPECT-GEARCASE-OIL-LEVEL-IDLE-WINCH
C23HZAN-TIME-RANGE) (359BZUN BT/EN/GSM/MM3 FLUSH-HELLAN-SEAWATER-STRAINER 359BZUN-TIME-
RANGE) (47K78GM BT/EN/MM3 LUBRICATE-PUMP 47K78GM-TIME-RANGE) (17A8UBN BT/MM3 INSPECT-
BUBLER-LIQUID-LEVEL 17A8UBN-TIME-RANGE) (64M44EN BT2 TEST-CLEAN-AND-INSPECT-FLAME-SCANNER
64M44EN-TIME-RANGE) (B2B7TCN DC/HT3 VISUALLY-INSPECT-PUMP-UNIT B2B7TCN-TIME-RANGE)
(B2A7SAN DC/HT3-1 VISUALLY-INSPECT-PUMP-UNIT-2 B2A7SAN-TIME-RANGE) (A4C2EHN DC/HT3-2 TEST-
AND-OPERATE-AFFF-CONCRETE-PUMP-ASSEMBLY A4C2EHN-TIME-RANGE) (34A1JWN DCPO INSPECT-
DIFFERENTIAL-PRESSURE/PRESURE-DROP 34A1JWN-TIME-RANGE) (36A1JVN DCPO-1 INSPECT-AND-TEST-
RELAY-OPERATED-LANTERN 36A1JVN-TIME-RANGE) (36W29WN DCPO-2 INVENTORY-AND-INSPECT-FIRE-
HOSE-STATION-EQMT 36W29WN-TIME-RANGE) (36W29XN DCPO-3 ACCOMPLISH-FUNCTIONAL-TEST-OF-
PORTABLE-LANTERN 36W29XN-TIME-RANGE) (36W31CN DCPO-4 INSPECT-AND-TEST-RELAY-OPERATED-
LANTERN2 36W31CN-TIME-RANGE) (266TFEN EM3-3 ACCOMPLISH-FUNCTIONAL-TEST-OF-ENGINE-BATTERY
266TFEN-TIME-RANGE) (51A1BNN EM3-2 INSPECT-MICROWAVE-OVEN 51A1BNN-TIME-RANGE) (628URAN
EM3-1 TEST-RUNNING-LIGHT-TELLTALE-PANEL 628URAN-TIME-RANGE) (482YTTN EM3 TEST-SIGNAL-AND-
NAVIGATION-LIGHTS 482YTTN-TIME-RANGE))

```

In order to fix the violated constraints we applied the P&R method from our library.

#### 8.4.4 Modelling the propose phase

The propose phase of the P&R method is configured straightforwardly from Generic-Schedule. The only main difference in the configuration process of Generic-Schedule and the propose phase is that the application-specific schedule-procedures (cf. Section 7.3.4.1) are defined in order to assign weekly ship-maintenance jobs to weekly ship-maintenance resources and time ranges. Also, the slot depends-on in the definition of schedule-procedure is instantiated in order to construct the dependencies between weekly ship-maintenance jobs. As an example the following box shows the OCML definition of the schedule procedures defined for 266TFEN job.

```

(def-class WEEKLY-RESOURCE-PROCEDURE (weekly-ship-resource-operator))

(def-class WEEKLY-TIME-RANGE-PROCEDURE (weekly-ship-time-range-operator))

(def-instance 266TFEN-JOB-TO-EM3-3-RESOURCE weekly-resource-procedure
  ((applicable-to-jobs `(setofall ?x (266TFEN-job ?x)))
   (depends-on `(51A1BNN)
    (has-body (lambda (?x ?s)
      (the ?EM3-3-RESOURCE
        (EM3-3-RESOURCE ?EM3-3-RESOURCE))))))

(def-instance 266TFEN-JOB-266TFEN-TIME-RANGE weekly-time-range-procedure
  ((applicable-to-jobs `(setofall ?x (266TFEN-job ?x)))
   (depends-on `(51A1BNN)
    (has-body (lambda (?x ?s)
      (the ?266TFEN-job-time-range
        (266TFEN-job-time-range ?266TFEN-job-time-range))))))

(tell (schedule-operator-order 266TFEN-job-to-EM3-3-resource
  266TFEN-job-266TFEN-time-range))

```

### 8.4.5 Modelling the fixes

In the revise phase application-specific fixes called ship-maintenance-time-range-fixes are defined in order to fix the constraint violations occurred while constructing a schedule. As described earlier the weekly ship-maintenance jobs, 482YTTN, 628URAN, 51A1BNN, and 266TFEN failed to comply with their latest end time, and the fixes are defined in such a way that these weekly ship-maintenance jobs can be shifted exactly by the same time (i.e. 10 minutes) by which they violated their latest end time. In scheduling this type of shift policy is referred to as the left-shift strategy (Smith, 1994). The following box shows the OCML definitions of two such fixes defined for the weekly ship-maintenance jobs 266TFEN and 628URAN. The fixes for the other jobs can be realised analogously.

```

(def-class SHIP-MAINTENANCE-TIME-RANGE-FIX (schedule-fix-for-time-range))

(def-instance 266TFEN-TO-NEW-TIME-RANGE ship-maintenance-time-range-fix
  ((applicable-to-jobs `(setofall ?x (266TFEN-job ?x)))
   (depends-on `(51A1BNN)
    (applicable-to-constraints `(DAILY-FREQUENCY-OF-266TFEN-job))
    (has-body (?x ?jtr ?sc)
      (cons ?x (and (has-time-range ?x ?266EFEN-job-time-range)
        (- (the ?let (= (latest-end-time-of-a-job
          ?x ?266EFEN-job-time-range) ?let))
          10))))))

(def-instance 628URAN-TO-NEW-TIME-RANGE ship-maintenance-time-range-fix
  ((applicable-to-jobs `(setofall ?x (628URAN-job ?x)))
   (depends-on `(482YTTN)
    (applicable-to-constraints `(end-time-compliance-of-628URAN))
    (has-body (?x ?jtr ?sc)
      (cons ?x (and (has-time-range ?x ?628URAN-job-time-range)
        (- (the ?let (= (latest-end-time-of-a-job
          ?x ?628URAN-job-time-range) ?let))
          10))))))

```

#### 8.4.5.1 Fix application in the revise phase

Once the configuration of the P&R method is completed then we again ran the weekly ship-maintenance application by using the P&R method. By the completion of the revise phase it was observed that our fixes successfully shifted all the weekly ship-maintenance jobs exactly by the same time towards their latest end time. As a result a complete and

consistent schedule for this application was constructed by generating 1401 schedule states. It was also observed that the precedence relation imposed among all the jobs also helped to maintain the daily frequency constraint imposed on the other weekly ship-maintenance jobs, which did not participate in the constraint violations. Therefore, a solution for this application was constructed linearly without any backtracking. However, on the negative side, the same precedence relation did not allow us to improve the overall cycle time of a schedule, because the earliest start time and the latest end time of each weekly ship-maintenance job was constrained by the earliest start time and the latest end time of the preceding weekly ship-maintenance job.

The following box represents the synoptic trace of the behaviour of the P&R method applied to the weekly ship-maintenance application. This trace particularly shows how all the constraint violations are collected as the foci in the revise context, and it also shows how the first constraint violation, i.e. `daily-frequency-of-482YTTN-job` from the list of foci is selected. Finally the box below also shows how a complete and consistent schedule looks like after fixing all the violations.

```

----- Enter task CONSISTENT-MAXIMAL-STATE-
SELECTION1253 with arguments (HAS-SCHEDULE-SPACE SCHEDULE-SPACE424)

----- Exit task CONSISTENT-MAXIMAL-STATE-
SELECTION1253 -> SCHEDULE-STATE1284

----- Enter task PROPOSE-AND-REVISE-CONTROL-
STRUCTURE1255 with arguments (HAS-SCHEDULE-SPACE SCHEDULE-SPACE424) (HAS-SCHEDULE-STATE
SCHEDULE-STATE1284)

----- Enter task ONE-STEP-REVISION-FOR-
CONSTRAINT1256 with arguments (HAS-SCHEDULE-SPACE SCHEDULE-SPACE424) (HAS-SCHEDULE-STATE
SCHEDULE-STATE1284)

----- Enter task GENERATE-NEW-STATE-SUCCESSOR1257
with arguments (HAS-SCHEDULE-STATE SCHEDULE-STATE1284) (HAS-SCHEDULE-CONTEXT :REVISE)

----- Enter task COLLECT-ALL-CONSTRAINT-
VIOLATIONS1258 with arguments (HAS-SCHEDULE-STATE SCHEDULE-STATE1284) (HAS-SCHEDULE-
CONTEXT :REVISE)

----- Exit task COLLECT-ALL-CONSTRAINT-
VIOLATIONS1258 -> (DAILY-FREQUENCY-OF-482YTTN-JOB DAILY-FREQUENCY-OF-628URAN-JOB DAILY-
FREQUENCY-OF-51A1BNN-JOB DAILY-FREQUENCY-OF-266TFEN-JOB)

----- Enter task PROPOSE-SCHEDULE-FROM-CONTEXT1260
with arguments (HAS-SCHEDULE-STATE SCHEDULE-STATE1206) (HAS-SCHEDULE-CONTEXT :REVISE)

----- Enter task SELECT-CANDIDATE-CONSTRAINT-
VIOLATION1262 with arguments (HAS-SCHEDULE-FOCI (DAILY-FREQUENCY-OF-482YTTN-JOB DAILY-
FREQUENCY-OF-628URAN-JOB DAILY-FREQUENCY-OF-51A1BNN-JOB DAILY-FREQUENCY-OF-266TFEN-JOB))
(HAS-SCHEDULE-FOCUS-ORDER-RELATION SCHEDULE-FOCUS-ORDER)

----- Exit task SELECT-CANDIDATE-CONSTRAINT-
VIOLATION1262 -> DAILY-FREQUENCY-OF-482YTTN-JOB

----- Enter task DEFAULT-SEARCH-CONTROL-RECORD-
ON-FOCUS-SELECTION-UPDATE1264 with arguments (HAS-SCHEDULE-FOCUS DAILY-FREQUENCY-OF-
482YTTN-JOB) (HAS-SEARCH-CONTROL-RECORD STATE-SEARCH-CONTROL-RECORD1259)

----- Exit task DEFAULT-SEARCH-CONTROL-RECORD-ON-
FOCUS-SELECTION-UPDATE1264 -> (HAS-SCHEDULE-FOCUS STATE-SEARCH-CONTROL-RECORD1259 DAILY-
FREQUENCY-OF-482YTTN-JOB)

----- Enter task COLLECTION-OF-APPLICABLE-
FIXES1266 with arguments (HAS-SCHEDULE-FOCUS DAILY-FREQUENCY-OF-482YTTN-JOB)

----- Exit task COLLECTION-OF-APPLICABLE-
FIXES1266 -> (482YTTN-TO-NEW-TIME-RANGE)

OCML 31 : 1 > (describe-instance 'schedule-state1401)

Instance SCHEDULE-STATE1401 of class SCHEDULE-STATE

HAS-SCHEDULE-MODEL: ((12B3HTN ABF/BT/EN2 TURN-PUMP-SHAFT-BY-HAND 12B3HTN-TIME-RANGE)
(63A2TFN ABF/EN/GSM/MM2 STRIP-JP5-SERVICE-TANK 63A2TFN-TIME-RANGE) (36A2RDN ABF/EN/MM3
INSPECT-SEAL-TANK-WATER-LEVEL 36A2RDN-TIME-RANGE) (44B9URN AN/FN/SN TEST-OPERATE-AND-
INSPECT-FLUSHOMETER 44B9URN-TIME-RANGE) (C23HZAN BM3 INSPECT-GEARCASE-OIL-LEVEL-IDLE-WINCH
C23HZAN-TIME-RANGE) (359BZUN BT/EN/GSM/MM3 FLUSH-HELLAN-SEAWATER-STRAINER 359BZUN-TIME-
RANGE) (47K78GM BT/EN/MM3 LUBRICATE-PUMP 47K78GM-TIME-RANGE) (17A8UBN BT/MM3 INSPECT-
BUBLER-LIQUID-LEVEL 17A8UBN-TIME-RANGE) (64M44EN BT2 TEST-CLEAN-AND-INSPECT-FLAME-SCANNER
64M44EN-TIME-RANGE) (B2B7TCN DC/HT3 VISUALLY-INSPECT-PUMP-UNIT B2B7TCN-TIME-RANGE)
(B2A7SAN DC/HT3-1 VISUALLY-INSPECT-PUMP-UNIT-2 B2A7SAN-TIME-RANGE) (A4C2EHN DC/HT3-2 TEST-
AND-OPERATE-AFFF-CONCRETE-PUMP-ASSEMBLY A4C2EHN-TIME-RANGE) (34A1JWN DCPO INSPECT-
DIFFERENTIAL-PRESSURE/PRESURE-DROP 34A1JWN-TIME-RANGE) (36A1JVN DCPO-1 INSPECT-AND-TEST-
RELAY-OPERATED-LANTERN 36A1JVN-TIME-RANGE) (36W29WN DCPO-2 INVENTORY-AND-INSPECT-FIRE-
HOSE-STATION-EQMT 36W29WN-TIME-RANGE) (36W29XN DCPO-3 ACCOMPLISH-FUNCTIONAL-TEST-OF-
PORTABLE-LANTERN 36W29XN-TIME-RANGE) (36W31CN DCPO-4 INSPECT-AND-TEST-RELAY-OPERATED-
LANTERN2 36W31CN-TIME-RANGE) (266TFEN EM3-3 ACCOMPLISH-FUNCTIONAL-TEST-OF-ENGINE-BATTERY
266TFEN-TIME-RANGE) (51A1BNN EM3-2 INSPECT-MICROWAVE-OVEN 51A1BNN-TIME-RANGE) (628URAN
EM3-1 TEST-RUNNING-LIGHT-TELLTALE-PANEL 628URAN-TIME-RANGE) (482YTTN EM3 TEST-SIGNAL-AND-
NAVIGATION-LIGHTS 482YTTN-TIME-RANGE))

```

## 8.5 The benchmark application

This is the last application used to validate our library. The data-set for this application was acquired from the following URL - <http://www.neosoft.com/~benchmr/rcps.doc>. This series of benchmark tests consists of twelve different applications. Although these applications are based on a large scale assembly, they can be applied to other scheduling domains, such as engineering, construction, and manufacturing. Generally speaking, this

application can be understood as a resource constrained project scheduling problem (Brucker *et al.*, 1999).

For validating our library, the application from the category 3 of this series was selected mainly because this application required using *looking ahead* heuristics while constructing a schedule. As described in Chapter 6 (cf. Section 6.3.1.1), Generic-Schedule deploys two types of looking ahead heuristics: *full looking ahead* and *partial looking ahead*, and this application provided an opportunity to evaluate the performance of these two heuristics.

This application consists of ten discrete work-steps (i.e., jobs that need to be executed to construct a large assembly) and each work-step entails the performance of a specific work document in the formal process plan. Each work-step has a fixed duration during which all the activities associated with a work-step have to be completed. The schedule horizon of one shift is 7.5 hours, which must be maintained by all the work-steps.

Each work-step requires either one or more resources (which are referred to as individuals in the data set) for its completion, which can be drawn from the four different labour pools. The labour pools are named anonymously as Px, Py, Pz, and Pw. While constructing a schedule, the appropriate labour types must be assigned for the successful completion of each work-step. There are thirteen different work-zones available around the assembly which are used by the work-steps for their completion. The available work-zones are named anonymously as Za, Zb, Zc, and so on. The execution of the work-steps depends on the availability of the work-zones, and therefore they are treated as a spatial type of resources. Moreover, all the work-steps must be completed from start to finish without any perturbation. Finally, this application imposes a strict precedence ordering among all the work-steps. Table 8.7 represents a data set obtained from the above mentioned URL, which is used to impose the precedence ordering among the work-steps.

Table 8.7. The precedence relation among work-steps.

Predecessor	Successor
asm_1.step_394	new.step_001
new.step_001	new.step_002
new.step_002	new.step_003
new.step_003	new.step_004
new.step_004	new.step_005
new.step_005	new.step_006
new.step_006	new.step_007

new.step_007	new.step_008
new.step_008	new.step_009
new.step_009	new.step_010
new.step_010	asm_1.step_518

Both the zone and labour resources have a fix capacity which determines the maximum number of work-steps they can handle at any one time. Table 8.8 represents the capacity of the zone and labour resources.

Table 8.8. The capacity of the zone and labour resources.

Assembly zone	Maximum capacity	Labour Pool	Maximum capacity
Zone.Za	2	Labour.Px	3
Zone.Zb	1	Labour.Py	4
Zone.Zc	1	Labour.Pz	4
Zone.Zd	2	Labour.Pw	5
Zone.Ze	1		
Zone.Zf	2		
Zone.Zg	1		
Zone.Zh	2		
Zone.Zi	5		
Zone.Zj	2		
Zone.Zk	1		
Zone.Zl	4		
Zone.Zm	3		

Table 8.9 shows the duration and a resource requirement of all the work-steps obtained from the above mentioned URL.

Table 8.9. The duration and resource requirement of all the work-steps.

Work-Step	Duration (hr:min)	P x	P y	P z	P w	Z a	Z b	Z c	Z d	Z e	Z f	Z g	Z h	Z i	Z j	Z k	Z l	Z m
new.step_001	03:30			2		1								1				
new.step_002	01:00			1						1								
new.step_003	02:30	1			1								1					
new.step_004	01:30		1					1										
new.step_005	00:30				2		1					1						
new.step_006	01:20			1	1							1		1				
new.step_007	03:04	1															1	
new.step_008	06:25			1			1											
new.step_009	00:10				1			1										
new.step_010	03:20		1										1					

8.5.1 Construction of a task model

In accordance with the task ontology the notion of a work-step is modelled by defining the application-specific class called assembly-job and this notion is mapped to the task level notion of a job (cf. Section 5.2.2). Each work-step has the following attributes: number of activities, resource requirement, a time range, duration, and a load. The following box shows the OCML definitions of the job work-step\_001-1.

```
(def-class new.step_001-1 (assembly-job))
(def-class new.step_001-1-TIME-RANGE (job-time-range))
(def-class new.step_001-1-ACTIVITY (activity))
(def-instance new.step_001-1 new.step_001-1-1
  ((has-activities new.step_001-1-activity)
   (requires-resource pz-resource)
   (has-time-range new.step_001-1-time-range)
   (has-duration new.step_001-1-duration)
   (has-load 2)))
```

Both the types of resources, labour and zone have the following attributes: specific types of jobs they can handle, availability period, and a capacity. The following box shows the OCML definitions of the labour resource called px-labour-resource and the zone-resource called za-zone-resource.

```

(def-class LABOUR-RESOURCE (resource))

(def-class ZONE-RESOURCE (resource))

(def-class px-labour-resource (labour-resource))

(def-instance PX-RESOURCE px-labour-resource
  ((has-job-belonging '(new.step_003-1 new.step_007-1))
   (has-availability resource-availability-period)
   (has-capacity 4)))

(def-class za-zone-resource (zone-resource))

(def-instance za-resource za-zone-resource
  ((has-job-belonging '(new.step_001-2))
   (has-availability resource-availability-period)
   (has-capacity 2)))

```

### 8.5.1.1 Modelling the constraints

The benchmark application includes the following two types of constraints:

- **The work-step precedence constraint:** This constraint is common to all the work-steps, and imposes a strict precedence ordering among all the work-steps. According to this constraint a work-step, say  $W_1$  precedes a work-step, say  $W_2$ , if the latest end time of  $W_1$  is before the earliest start time of  $W_2$ . This precedence constraint is imposed by using the data given in Table 8.7. A set of ten precedence constraints are defined between the work-steps. The following box shows the OCML definition of one such constraint imposed among the work-steps 'new-step\_001\_1' and 'new-step\_001\_2'.

```

(def-instance PRECEDNCE-AMONG-NEW-STEP_001_1-NEW-STEP_001_2
  assembly-job-constraint
  ((has-expression (kappa (?sc)
    (exists ?new-step_001_1
      (and (has-time-range
        ?new-step_001_1
        ?new-step_001-1-time-range)
        (has-latest-end-time
        ?new-step_001-1-time-range ?let)
        (exists ?new-step_001_2
          (and
            (has-time-range
              ?new-step_001_2
              ?new-step_001-2-time-range)
            (has-earliest-start-time
              ?new-step_001-2-time-range ?est)
            (precedes ?let ?est)
            (JOB-PRECEDES
              ?new-step_001_1
              ?new-step_001_2))))))))))

```

- **The labour and zone resource constraint:** The labour capacity constraint imposes a restriction on the number of work-steps each labour and zone resource can handle at any one time while constructing a schedule. This constraint is imposed by using the resource capacity data given in Table 8.8. A set of four resource constraints are defined to impose capacity constraint on the four labour resources and a set of thirteen constraints are defined to impose the capacity constraint on all the zone resources. In order to impose this constraint on the labour and zone resources, we first used the function called maximum-capacity-of-resource (cf. Section

5.2.3), which retrieves the maximum number of work-steps each labour and zone resource can handle, and then an equality condition is imposed to constrain the maximum number of work-steps each resource can handle according to the data given in Table 8.8. The following box shows the OCML definitions of the capacity constraints imposed on the labour resource ‘px-labour-resource’ and the work-zone resource ‘za-zone-resource’.

```
(def-class ASSEMBLY-LABOUR-CONSTRAINT (hard-constraint))

(def-class ASSEMBLY-ZONE-CONSTRAINT (hard-constraint))

(def-instance PX-LABOUR-CAPACITY assembly-labour-constraint
  ((applicable-to-resources `(setofall ?x (px-labour-resource ?x)))
   (has-body (lambda (?x ?sc)
                (exists ?x
                  (= (the ?x1 (maximum-capacity-of-resource ?x))
                     3))))))

(def-instance ZA-ZONE-CAPACITY assembly-zone-constraint
  ((applicable-to-resources `(setofall ?x (za-zone-resource)))
   (has-body (lambda (?x ?sc)
                (exists ?x
                  (= (the ?x1 (maximum-capacity-of-resource ?x))
                     2))))))
```

While formalising this application, no additional definitions were needed in addition to those already exists in the scheduling task ontology. All the key classes from the task ontology such as job, resource, activity, time range, etc. provide the required level of detail and precision to capture the application-specific knowledge precisely. In the following section, we will discuss how the schedule for this application was constructed.

### 8.5.2 Applying the Propose & Backtrack method

To configure the Propose & Backtrack method, two types of application-specific operators, `benchmark-resource-operator` and `benchmark-time-range-operator` were defined. The former type of operator is used to assign the work-steps to their required zone and labour resources as given in Table 8.8. The latter type of operator is defined in such a way that the correct time range can be assigned to all the work-steps for their in time completion. In total ten operators were defined. The following box shows the OCML definitions of the operators defined for work-step `new-step_001-1`.

```

(def-class benchmark-resource-operator (schedule-extension-resource-operator))

(def-class benchmark-time-range-operator
  (schedule-extension-time-range-operator))

(def-instance new-step-001-1-to-pz-labour-resource benchmark-resource-operator
  ((applicable-to-jobs '(setofall ?x (new-step_001_1 ?x)))
   (has-body (lambda (?x ?sc)
                (the ?pz-resource
                    (pz-resource ?pz-resource
                                has-job-belonging ?x))))))

(def-instance new-step-001-1-to-new-step-001-1-time-range benchmark-time-range-operator
  ((applicable-to-jobs '(setofall ?x (new-step_001_1 ?x)))
   (has-body (lambda (?x ?sc)
                (the ?new-step_001-1-time-range
                    (new-step_001-1-time-range ?new-step_001-1-time-range))))))

(tell (schedule-operator-order new-step-001-1-to-pz-labour-resource
                              new-step-001-1-to-new-step-001-1-time-range))

```

### 8.5.2.1 Focus and operator selection

As described earlier this application imposed a strict precedence among all the work-steps and while selecting a correct focus we complied with this application-specific knowledge. The method called *job-selection-based-on-precedence* (cf. Section 6.3.5) is used to select the focus. To represent the precedence among different work-steps we instantiated the relation *job-precedes* (cf. Section 5.2.2.3) from the scheduling task ontology and then this relation is used by the method *job-selection-based-on-precedence* to sort the work-steps. Once a correct focus is selected then the order of operator application is determined by instantiating the relation *schedule-operator-order*.

### 8.5.2.2 Analysis

Having configured the P&B method, we first ran our experiment focusing on the full looking ahead heuristic. The basic idea of this heuristic is that when a value is assigned to a variable the problem is reduced through constraint propagation.

Because this application imposed a tight precedence constraint on all the work-steps, the solution space of this application was very well structured. This helped to improve the performance of the full looking ahead heuristic because this heuristic usually performs well on problems with tight constraints. However, while constructing a schedule by using full looking heuristic it was realised that this heuristic was computationally very expensive. In comparison with the partial looking ahead heuristic this heuristic required almost one hundred more schedule states to reach a solution state. One of the main reasons for the computational cost of this heuristic was that each time a new work-step was selected for its assignment this heuristic imposed a full consistency check of the value requirement (i.e., resources and time ranges) of the current work-step. In other words, the system was performing a value requirement consistency between a currently instantiated work-step and other unassigned work-steps, as well as between all unassigned work-steps. It was

observed that these checks did not discover any new inconsistencies often enough to justify the large number of consistency checks performed. On the positive side, such type of consistency checks removed all those values from the domain of the future work-steps which were not compatible with the current assignment. This essentially helped to detect all dead-ends beforehand and therefore the search reached the solution state without any backtracking. As a result, 100% efficiency was achieved while constructing a schedule. The complete schedule for this application was constructed by generating 805 schedule states.

Then we ran the same application by using the partial looking ahead heuristic to analyse the performance of this heuristic. In comparison with the full looking ahead heuristic, the partial looking ahead heuristic proved to be computationally more efficient. The main reason why the partial looking ahead heuristic took less time to reach the solution state was because it made about half the consistency checks as compared to the full looking ahead heuristic. The partial looking ahead heuristic checked the value requirement consistency between the current work-step with all the *unassigned* work-steps, which directly or indirectly depend on it. As a result, by using the partial looking ahead heuristic, the complete schedule was constructed by generating 705 schedule states. Although this heuristic reached a solution state much more quickly compared to the full looking ahead one, the conflict detection policy of the full looking ahead heuristic is more robust and exhaustive, and has a higher chances of avoiding conflicts between the assigned and unassigned jobs.

Finally, our focus selection strategy helped to accomplish all the work-step jobs within their single shift. Because no constraints and requirements were violated while constructing a schedule, a complete and consistent schedule was returned.

Table 8.10 summarises the performance of our library on all the applications that are used to validate our library.

Table 8.10. Comparison between the performances of scheduling applications.

Application name	Problem-solving method used to solve the application	Job-selection heuristic used to select a job	Number of schedule states required to generate a solution schedule
The satellite-scheduling	Propose & Backtrack, Hill	Job-selection-based-on-	464

application	Climbing, Propose & Improve	lowest-degrees- of-freedom	512
CIPHER- a resource allocation application	Propose & Backtrack	Job-selection- based-on- latest-end-time	1342
The daily ship- maintenance application	Generic- Schedule,  Propose & Restore- feasibility	Job-selection- based-on-start- time	852  949
The weekly ship- maintenance application	Propose & Backtrack,  Propose & Revise	Job-selection- based-on-least- number-of- activities	1245  1401
The benchmark application	Propose & Backtrack	Job-selection- based-on- precedence	805  705

## 8.6 Evaluating the static and dynamic properties

As described by Preece *et al.* (1996), when we consider the validation and verification problem, it is useful to distinguish between the static and dynamic properties of a rule-based system. The static properties are those characteristics of a rule-based system that can be evaluated without its execution, while the dynamic properties can be evaluated only by examining how the system operates at run time. The following bullet points describe the different characteristics that we have validated to evaluate the performance of our library.

- The goal requirements of each application that needs to be achieved by the PSMs in our library;
- The quality of the goal specifies whether a selected PSM from our library has successfully achieved the goals specified by the application;
- The relation between the application-specific data and the way it has influenced to achieve the goals proposed by the applications.

Table 8.11 summarises the results of our study.

Table 8.11. Summary of the evaluation of the static and dynamic properties.

Application name	Goal requirements	Quality of the goal	Relation between data and goals achieved
The satellite-scheduling application	To generate a complete and optimal schedule.	The P&B method has successfully constructed a complete solution schedule, but it was not an optimal one.	The data of this application was good enough to construct a complete schedule, but while using the hill climbing method it did not provide enough discriminating knowledge to break the tie between the assignments of Nimbus-1 and Nimbus-2 satellites. However, while using P&I, the data allowed us to swap the time slots of these two satellites to generate an optimal assignment.
		The hill climbing method not only failed to produce an optimal solution, but it took higher number of schedule states to generate a complete schedule.	
		The P&I method successfully devised a complete and optimal schedule.	
CIPHER - a resource allocation application	To generate a complete schedule.	The P&B method has successfully constructed a complete schedule.	The data provided by this application was well specified. Therefore, solving this application by using P & B turned out to be efficient as the solution space was very

			dense. As a result, very little search was required to construct a complete solution schedule.
The daily ship-maintenance application	To generate a complete and feasible schedule.	Generic-Schedule has successfully devised a complete schedule, but it was not a feasible one.	The data provided by this application was contradictory in nature. This application required all the jobs to complete within their daily frequency, which led us to use a job selection heuristic that selected the jobs based on their earliest start time. However, this job selection violated a 'job priority requirement', which essentially needed to give priority to those jobs which have higher number of activities for their selection.
		The P&Rf method has successfully devised a feasible schedule by fixing the requirement violation.	
The weekly ship-maintenance application	To generate a complete and consistent schedule.	The P&B method has successfully devised a complete schedule, but it was not a consistent one.	1) In compliance with the data, a job with the least number of activities

		<p>The P&amp;R method was successfully devised a consistent schedule by fixing all the constraint violations.</p>	<p>was selected as a focus in P&amp;B, but it violated the 'daily frequency of ship-maintenance job' constraint.</p> <p>2) A strict precedence constraint imposed by the application data allowed us to maintain the daily frequency constraint of other jobs that were not part of the constraint violation, but it did not allow us to improve the overall cycle time.</p>
The benchmark application	To generate a complete schedule	The P&B method has successfully devised a complete schedule.	The data of this application was very well structured, which helped to improve the performance of the full looking ahead heuristic, as this heuristic performs well on problems with tight constraints.

## 8.7 Conclusion

In this chapter we have described the validation study of our library, which has been carried out on a number of scheduling applications. The applications used to validate our

library covered a wide range of scheduling domains, such as space scheduling, resource allocation, and manufacturing. Despite the fact that these applications come from different domains, the methods in our library performed successfully.

As described in Section 1.4, because our task ontology formalised the scheduling task without subscribing to any application domain of scheduling, it allowed us to formalise all the heterogeneous scheduling applications successfully. In contrast with our task ontology some of the existing task ontologies (Hama *et al.*, 1992a, b and Smith and Becker, 1997) subscribed to the specific domain of scheduling and therefore it is difficult to realise how these task ontologies could have formalised the scheduling applications coming from different domains. Moreover, the existing scheduling task ontologies (Hama *et al.*, 1992a, b; Mizoguchi *et al.*, 1995; Smith and Becker, 1997) have only provided an incomplete coverage to the different concepts necessary to characterise the scheduling task. For instance, the MULTIS task ontology (Mizoguchi *et al.*, 1995) failed to take into account a crucial concept like resource-capacity (cf. Section 5.2.3.1), and therefore this task ontology would have failed to provide a support to avoid job overlapping in the CIPHER application. Also in contrast with the job assignment task ontology (cf. Sections 3.4.2.1, 5.3.1), all the concepts from our scheduling task ontology have provided an appropriate level of detail to formalise the application-specific knowledge. Finally, because our task ontology has provided an unequivocal distinction between constraints, requirements, and preferences, it helped us to formulate the application-specific knowledge without having to compromise with their meaning as shown in all the applications.

At the problem-solving level, the search acted as a fundamental problem-solving paradigm, which enabled a strong coupling<sup>2</sup> between the scheduling task specification and the method specification. As described in Section 1.4, Generic-Schedule subscribed to the knowledge-intensive approach to schedule construction, which has abstracted different tasks, methods, heuristics, and also taken into account the domain-specific knowledge. And as shown throughout this chapter, these tasks and methods were reused to effectively to reason about different scheduling applications. Moreover, different heuristics from Generic-Schedule, such as *dynamic consistency enforcement*, *full looking ahead* and

---

<sup>2</sup> In our library, the problem-solving methods are developed to perform an efficient problem-solving to solve a specific type of generic task, i.e. scheduling. This close association between a generic task specification and a method represents a strong coupling of the library. This coupling can be further strengthened by using a choice of a problem-solving paradigm (which is search in our library) as a mechanism for providing a principled approach for developing a generic problem-solving model and a method ontology for a given problem type.

*partial looking ahead* improved the overall efficiency of schedule construction. Also, in contrast with the existing libraries (Hori and Yoshida, 1998; Sundin, 1994; Tijerino and Mizoguchi, 1993; Le Pape, 1994), which have provided only a limited support for a job selection, our library have provided a wide-range of job selection heuristics (cf. Section 6.3.5), which not only selected a focus correctly, but also avoided unnecessary backtracking while constructing a schedule in all the applications.

Because our library consisted of a wide-range of PSMs, it allowed us to tackle the different types of inconsistencies, such as constraint or requirement violations, which occurred in different applications. In contrast with the comprehensive coverage provided by our library, none of the existing libraries (Hori and Yoshida, 1998; Sundin, 1994; Tijerino and Mizoguchi, 1993; Le Pape, 1994) have included the problem-solving methods which can reason about the requirement violation and optimisation issues of scheduling. Moreover, because our library did not subscribe to any scheduling domain all the PSMs were reused to construct heterogeneous applications with either very little or in some cases no configuration effort.

# Chapter 9

## SUMMARY AND CONCLUDING REMARKS

In this chapter, we conclude our work by summarising the research carried out in this thesis, highlighting the main contributions of this work and suggesting future research directions.

### 9.1 Summary

In this thesis, we have proposed a generic library of scheduling problem-solving methods. Our library subscribes to the TMDA knowledge modelling framework (Motta, 1999), which provides the key epistemological distinctions required to model scheduling engineering knowledge-based applications by reuse.

In compliance with the TMDA framework, we first formalised the space of scheduling problems by developing a generic task ontology (cf. Chapter 5). The task ontology is generic in the sense that it does not subscribe to any application domain or problem solving method. Then at the method level, we proposed a generic problem solving model, *Generic-Schedule* component of the library (cf. Chapter 6), which provides a comprehensive collection of tasks and methods, which cover the space of knowledge-based activities carried out during scheduling problem-solving. These tasks and methods can be specialised to construct more specific scheduling problem-solvers. As described in Chapter 7, seven different knowledge-intensive methods, Hill Climbing, Propose & Backtrack (Runkel *et al.*, 1996), Propose & Improve (Motta, 1999), Propose & Revise (Marcus and McDermott, 1989), Propose & Restore-feasibility, Propose & Exchange (Poeck and Gappa, 1993), and Propose & Genetical-Exchange were constructed by reusing and specialising the tasks defined in *Generic-Schedule*. This uniform approach to method construction allowed us to compare and contrast the knowledge requirements of these PSMs. Moreover, these PSMs cover a wide range of scheduling task specifications with respect to criteria such as completion, constraint violation, requirement violation, and optimisation. Finally, as described in Chapter 8, our library has been validated on a number of scheduling applications, which confirmed its generic nature.

Our work contributes to scheduling research both from an analytical and an engineering perspective. Analytically, it provides both a novel integration of the various techniques that have been developed for scheduling and provides an insight into the various knowledge-intensive tasks that are carried out during scheduling problem solving. From an engineering perspective, our library offers comprehensive support for the rapid

construction of scheduling applications in different domains. Finally, ours is the first library in the field that provides a comprehensive coverage of a variety of knowledge-intensive PSMs.

In the following section we discuss the major contributions of our research.

## 9.2 Contributions

### 9.2.1 A generic scheduling task ontology

As discussed in Chapter 3 (cf. Section 3.4.2) existing scheduling task ontologies - the job-assignment task ontology (Hori *et al.*, 1995; Hama *et al.*, 1993a, b), MULTIS (Mizoguchi *et al.*, 1995), and OZONE (Smith and Becker, 1997), have provided limited results. In some cases (Hama *et al.*, 1992a, b; Smith and Becker, 1997) these proposals focused on a specific scheduling domain, which restricted their reusability. In contrast with such domain-specific approaches, our task ontology formalises the scheduling task without subscribing to any specific domain, and therefore, it provides wider coverage and better support for application development by reuse. Other task ontologies (Smith and Becker, 1997) subscribed to a specific ‘problem-solving shell’. As a result, they only cover a subset of the space of scheduling tasks. In contrast with these approaches, our task ontology is independent of any specific problem-solving shell, and therefore, the concepts from our task ontology can be mapped to different problem-solving shells, tackling different types of scheduling tasks. Moreover, as described in Chapter 3 (cf. Section 3.4.2.4) existing task ontologies fail to address some of the important concepts that are necessary to characterise the scheduling task precisely. In particular, concepts such as requirements and preferences are typically missing. In contrast with these proposals, our task ontology provides a more sophisticated set of ontological distinctions separating constraints from requirements and preferences. The utility of these distinctions was shown in Chapter 8, where we demonstrated the importance of these distinctions for a correct modelling of task knowledge. Moreover, as shown in Chapter 8, the practical contribution of our task ontology is that it can be used as an ‘off the shelf’ resource to perform knowledge acquisition and formalise scheduling knowledge.

### 9.2.2 A generic model of scheduling problem solving

One of the main limitations of the existing scheduling libraries (Hori and Yoshida, 1998; Sundin, 1994; Le Pape, 1994; Tijerino and Mizoguchi, 1993) is that these proposals fail to provide a clean distinction between highly reusable generic components and non-reusable components. Therefore, it becomes very difficult to realise how the different components from these libraries can be reused to construct new PSMs. In contrast with the existing

proposals, our generic problem solving model, Generic-Schedule provides a comprehensive and generic framework, which can be easily specialised to produce different PSMs. At the same time, these tasks and methods provide an insight into the various knowledge-intensive activities that take place during scheduling problem solving. Because a component like Generic-Schedule is missing from existing proposals (Hori and Yoshida, 1998; Sundin, 1994; Le Pape, 1994; Tijerino and Mizoguchi, 1993), these fail to offer the same degree of reusability.

Another important contribution at the method level was provided by the specification of a generic method ontology. This method ontology offers a highly generic vocabulary to characterise the search-based problem-solving behaviour of our scheduling PSMs.

Moreover, Generic-Schedule exhibits a nice integration of the results from the constraint satisfaction community. For instance, heuristics such as *downstream consistency enforcement* (Sadeh, 1994), *full looking ahead*, *partial looking ahead* (Haralick and Elliot, 1980) are included in Generic-Schedule. From a search perspective, Generic-Schedule proposes a wide range of job selection methods (cf. Section 6.3.5), which can improve the efficiency of schedule construction by reducing unnecessary backtracking. Finally, Generic-Schedule itself can be used as a reusable and operational scheduling component to construct scheduling applications.

### 9.2.3 A comprehensive repertoire of scheduling problem solvers

Our library improves existing proposals (Hori and Yoshida, 1998; Sundin, 1994; Le Pape, 1994; Tijerino and Mizoguchi, 1993) in terms of three dimensions: *size of the library*, *coverage of the PSMs with respect to different types of scheduling problems*, and *reusability*. In contrast with the existing libraries (Hori and Yoshida, 1998; Sundin, 1994; Le Pape, 1994; Tijerino and Mizoguchi, 1993), our library provides a comprehensive and rich repertoire of the knowledge-intensive PSMs to tackle the scheduling task. For instance, the CommonKADS library (Sundin, 1994) only takes into account the Propose & Revise method (Marcus and McDermott, 1989), while other libraries (Hori and Yoshida, 1998; Le Pape, 1994; Tijerino and Mizoguchi, 1993) provide very limited set of PSMs. In addition our PSMs are very heterogeneous dealing with constraint and requirement violations as well as schedule optimisation issues. Because existing libraries provide only a limited set of scheduling PSMs, they fail to tackle the different types of scheduling problems. Finally, some of the existing libraries (Hori and Yoshida, 1998) tackle the scheduling task only from the perspective of a specific domain, such as production scheduling, and therefore, they have limited reusability. As shown in Chapter 8, the

domain independent nature of our library allows us to solve scheduling applications in different domains.

#### 9.2.4 Contribution to scheduling knowledge acquisition

Throughout the construction of our library, we have developed different types of templates either to construct the ontologies or to compare and contrast the knowledge requirements of different PSMs (cf. Section 7.1). These generic templates and the ontologies can be used to acquire the relevant scheduling knowledge. Here, the term ‘knowledge acquisition’ can be understood both in analytical and practical terms, given that this acquired knowledge is directly used to obtain concrete problem solvers.

#### 9.2.5 Contribution to scheduling epistemology

Our scheduling task ontology is based on a clear theoretical model of the scheduling task (cf. Section 5.1), which distinguishes between different components such as constraints, requirements, and preferences. Moreover it also provides an adequate level of detail to specify all the components necessary to characterise a scheduling problem. As a result, it acts as a clear reference point to frame the space of scheduling problems.

At the method level, Generic-Schedule and other PSMs in our library provide a theoretical insight into the various knowledge-intensive activities needed for constructing a schedule.

#### 9.2.6 Development of job selection heuristics

Another contribution made by this thesis to the scheduling domain is provided by the three job selection heuristics (cf. Section 6.3.5). As discussed in Chapter 2 (cf. Section 2.6), several rules and heuristics have been developed both in OR and AI to select a correct job. The selection of a correct job is an important activity in scheduling because it improves the efficiency of the schedule construction process. These heuristics are as follows:

- a) Job-selection-based-on-due-date: *if any two jobs are competing with each other for the usage of the same resource, then a job with the earliest due date is always given priority for its execution.* Panwalkar and Iskander (1977) list more than one hundred job selection rules and one of the rules from their list selects a job based on a due date. The fundamental difference between their rule and our heuristic is that, in our heuristic a job with the earliest due date is selected only when this job is competing with some other jobs for the same resource, while no such condition is imposed in their rule;
- b) Job-selection-based-on-bottleneck-resources: *the jobs that are using the bottleneck resources are always given priority.* Such jobs are assumed to provide better control in maintaining the global stability of a schedule;

c) Job-selection-based-on-number-of-activities: *a job with the highest number of activities is given priority.*

These heuristics are particularly important as they reduce unnecessary backtracking during schedule construction by selecting a correct job.

Having described the main contributions of our research, in the following section we will discuss future research directions.

### 9.3 Future research directions

#### 9.3.1 Extending the current technology to develop a planning library

Our existing technology can be extended to address the planning domain. Like scheduling, planning can be seen as a synthesis task, which involves formulating a sequence of actions to achieve a desired goal. Although, at a theoretical-level the planning and scheduling tasks can be distinguished on the basis of their goal criteria, in real-life this distinction often gets blurred. The planning task determines how the actions can be sequenced to achieve the desired goal, whereas the scheduling task allocates these actions on the available resources within a specific time range. Over the years, various planning paradigms have emerged in AI, such as Classical Planning (Fikes and Nilson, 1971), Decision Theoretic Planning (Blythe, 1999), and Hierarchical Task Network (HTN) (Erol *et al.*, 1994). However, as pointed out by Smith *et al.* (2000) all the planning systems which have been developed for the practical applications subscribe to the HTN planning paradigm (Nau *et al.*, 1999).

Some attempts have been made in the past at developing a library of PSMs to solve the planning task (Blythe and Gil, 1999; Valente *et al.*, 1998; Tu and Musen, 1996). These libraries can be taken as a starting point for our research and our ultimate aim is to amalgamate the different planning paradigms to construct a truly generic planning system.

#### 9.3.2 Interactive scheduling component

Because our library uses different specialised knowledge modelling techniques, the existing version of our system requires a certain level of expertise from its end users to produce scheduling applications. For instance, a scheduling application needs to be formalised by constructing its application ontology and a user needs to have enough knowledge to formalise his/her application by using the appropriate knowledge modelling language. As a result, these are high-end technological barriers for any non-technical users who wish to use our library. In future, we aim to lower this technical barrier by developing an *interactive* component to facilitate the use of our library.

The following points describe the main features that will be included in our envisaged interactive scheduling component:

- 1) **Construction of the KA forms:** The main purpose behind the development of the KA forms is to accelerate the process of representing different types of applications. The user should be able to represent his/her applications simply by populating the slots of the existing classes by using the KA forms. Moreover, we also envisage a certain level of flexibility that will enable end users to change other properties of the existing definitions. Having represented an application, a user can simply select one of the existing PSMs from our library to construct a solution schedule. At this stage we would like to take advantage of some of the existing in-house technologies, such as IRS-II (Motta *et al.*, 2003), which offers these types of functionalities;
- 2) **Schedule manipulation in a semi-automatic or manual mode:** This would allow a human scheduler to interact with the system to update the status of an existing schedule in compliance with the dynamic changes that occur in the scheduling environment. This mode is aimed to provide a scheduler with an overview of a schedule at any given time by displaying the current allocation status of the resources in a clear and systematic manner. Furthermore, a scheduler should also be able to enter new changes in the existing schedule such as interrupting a job during its execution, starting a job somewhere else on a time-line, or changing a sequence of job executions to improve the performance of a schedule. The remaining part of a schedule should adapt dynamically according to the changes introduced by a scheduler;
- 3) **Schedule representation in familiar formats:** Once a schedule is constructed then this component is expected to supervise a scheduler by displaying the status of a schedule and job processing in terms of a Bar Chart. According to the status and urgency of the jobs they cover be displayed by using different colour schema. Based on the status of the jobs a scheduler can then enter into the semiautomatic or manual mode to update a schedule. Due to this type of continuous feedback from the system, a scheduler is expected to gain more control over the entire scheduling process. Finally, a solution schedule can be represented by using familiar formats, such as Gantt Charts.

### 9.3.3 Towards Nano-Planning

Nanotechnology proposes a fundamental breakthrough to both biological and non-biological problems. The idea of atomic scale engineering originated from the cornerstone

talk given by Richard Feynman on December 29<sup>th</sup> 1959<sup>1</sup> at the annual meeting of the American Physical Society. The notion of atomic engineering resides at the core of nano-scale engineering whereby the atoms themselves can be seen as pre-fabricated components (Merkle, 1997). Today's manufacturing process is rather crude at the atomic level, but according to the envisaged vision of nanotechnology in the future it will become possible to arrange these primary building-blocks precisely in accordance with the laws of physics. The environment in which this process takes place is referred to as the *eutectic environment* (Drexler, 1992). The cost effectiveness of this process is a crucial factor that can be achieved by automating the molecular manufacturing process (Drexler, 1992). The engineering process of the molecular size products can be achieved by nano-scale robots, which are referred to as the *assemblers* (Merkle, 1996).

The planning and scheduling paradigms<sup>2</sup> will be key methods to determine the atomic assembly sequences in the *eutectic environment* (Drexler, 1992; Kandikjan and Dukovski, 1995). The *automated planning* in particular can be envisaged to be crucial for the development of the nano-scale components and has the following components - knowledge representation and domain modelling, traditional planning, scheduling and constraint satisfaction, machine learning and adaptive planning, nano-robotic fine motor control, computer-aided economic analysis, and advanced graphical simulation. Along with the planning techniques, the issue of time optimal schedules will be particularly important in the molecular manufacturing process, because time will be one of the major determinants for the cost-effectiveness of the molecular engineered components. Finally, different techniques from the constraint satisfaction literature will be particularly important when the manufacturing is carried out interactively by a large group of assemblers, given that it will be essential to optimise the organisation of these large groups of assemblers, each of which subject to energy and spatial constraints.

In sum it is likely that, the development of efficient and cost-effective assemblers will bring a revolution in various industrial sectors, such as medicine, manufacturing, energy efficient processes, space and aeronautical research. Our library provides an initial component of the technology required to achieve this vision.

---

<sup>1</sup> This talk can be found on the following URL: <http://www.zyvex.com/nanotech/feynman.html>

<sup>2</sup> (Czarn and MacNish: <http://citeseer.nj.nec.com/104691.html>)

# Glossary

## **Reusability:**

Our library is generic in the sense that it does not subscribe to any specific domain of scheduling, while our task ontology is reusable because it formalises the scheduling task without subscribing to any application domain of scheduling or the way scheduling problem can be solved.

## **Ontology:**

*An ontology is an explicit specification of a conceptualisation.* It provides a shared and common understanding of a domain that can be communicated across people and computation systems dealing with applications within a specific domain.

## **Problem-solving methods (PSM):**

A PSM describes the inference process underlying a KBS in an *implementation* and *domain-independent* way.

## **Task/Generic Task:**

The notion of a task specifies a goal for a problem solver, such as producing a valid schedule for the satellite-antenna communication. The notion of task is crucial to knowledge modeling because the knowledge-based systems are characterised and evaluated on task-specific criteria.

The notion of a Generic Task specifies a knowledge level, application-independent description of a goal, which has to be achieved by the problem solver.

## **Knowledge modelling approach:**

The notion of a knowledge modelling approach can be understood as follows: 1) knowledge engineering is not a cognitive modelling, i.e. reproducing expert reasoning, but it is about developing systems that perform knowledge-based problem solving and its performance can be evaluated in a task-specific way, 2) heterogeneous classes of applications has similar features that can be reasoned about by constructing generic models of problem solving, 3) the process of knowledge acquisition should not be characterised as a process of mapping expert knowledge to a computational representation, but it is an intelligent model-building process in which the application specific knowledge is configured according to available problem-solving technique, and 4) such intelligent

models can be described at a level, which abstracts from implementation considerations.

**Knowledge acquisition approach:**

The notion of knowledge acquisition can be realised by the following two ways: knowledge acquisition as mining and knowledge acquisition as a modelling. The 'knowledge acquisition as mining' can be characterised in terms of the earlier expert systems, which refers to the fact that discrete and distinct expertise knowledge can be elicited systematically from the domain experts.

In this thesis we subscribe to the 'knowledge acquisition as a modelling' approach. The crucial features of this approach can be realised based on knowledge modelling approach described earlier.

**Control regime:**

Modelling the problem-solving behaviour involves more than making statements and describing entities in the world. Control regimes are required to specify actions and describe the order in which these are executed. OCML supports the specification of sequential, iterative, and conditional control structures by means of a number of control term constructors such as *repeat*, *loop*, *do*, *if* and *cond*, among others.

**Slot:**

The slots, say  $s_{c1}$  of a class, say  $c_1$  has a unique binding with  $c_1$ , which represent the attributes of  $c_1$ . For instance, if there is a class called *job*, which has attributes, such as it requires certain resources for its execution, has a time range within which a job must be accomplished, etc. then these attributes can be represented by the following slots - *requires-resource* and *has-time-range*.

**Context:**

The notion of a context specifies the primary function of the problem-solving method within each problem-solving phase that has to be carried out to achieve a solution. For instance, the *Propose & Exchange* method (Poeck and Puppe, 1992) has two problem-solving phases - the *propose* phase and the *exchange* phase, and the context in the *propose* phase is to extend an incomplete schedule such that a complete solution schedule is generated, whereas if any of the constraints are violated while

constructing a complete schedule then the context in the exchange phase is to revise an inconsistent schedule to fix the constraint violations.

### **Focus:**

The notion of a focus exemplifies those variables in the problem formulation which are under scrutiny during each phase of the problem-solving method and these variables must be grounded in order to construct a solution. For instance, the Propose & Exchange method (Poeck and Puppe, 1992) has the following two problem-solving phases: the propose phase and the exchange phase. Because the main function of the propose phase is to construct a complete schedule, and therefore, in this phase the focus is on one of the unassigned jobs, which must be assigned to construct a complete solution schedule. And if any of the constraints are violated while constructing a complete schedule then the exchange phase is invoked to fix the constraint violations and the focus in this phase on those constraint violations which must be fixed.

### **Knowledge-roles:**

In compliance with the Generic Tasks approach (Chandrasekaran, 1986) a top-level task (which in our case is the scheduling task) can be decomposed into a small number of sub-tasks and sub-methods can be proposed to achieve these tasks. These tasks specify the application domain specific static and dynamic knowledge for their execution. These knowledge pieces essentially represent the abstract names of data objects that represent the role of these objects in the reasoning steps.

### **Strong and weak coupling:**

In our library the problem-solving methods are developed to perform an efficient problem-solving to solve the specific type of generic task, i.e. scheduling. This close association between a generic task specification and a method represents a strong coupling of the library. This coupling can be further strengthened by using a choice of a problem-solving paradigm (which is search in our library) as a mechanism for providing a principled approach to developing a generic problem-solving model and a method ontology for a given problem type.

In our library the domain-specific knowledge is multi-functional in nature such that this knowledge characterizes the task independent aspects of a domain. As a result, this domain-specific knowledge can be used in many

ways. However, this domain-specific knowledge can be used in order to improve the efficiency of problem-solving, and this association between multi-functional domain knowledge and its utilization within problem-solving method represents a weak coupling.

**Knowledge-intensive PSMs:**

Knowledge-intensive problem-solving methods are the ones that make heavy use of the application domain specific knowledge during problem solving. For instance, in our library the operators that are used to assign jobs to resources and time ranges are constructed and selected in compliance with the application specific knowledge. Different job selection heuristics used to select a correct job make effective use of the domain specific knowledge while executing this problem-solving action.

**Constraint and requirement:**

Constraints represent those properties which a solution schedule must not violate under any circumstances throughout a schedule construction.

Requirements represent those properties which a solution schedule should satisfy in order to become a feasible solution.

**Precondition:**

The preconditions are associated with a goal-specification task and they are used to specify what must be true before executing a goal-specification task.

**Goal-expression:**

The goal-expression is used to specify the goal associated with a goal-specification-task. For instance, in our library the scheduling task is a goal-specification task and a goal associated with this task is to generate a valid schedule.

**Job/activities:**

The notion of a job represents an entity that has a list of activities and can be assigned over available resources and time ranges for its execution.

Each job can have a list of activities that need to be performed in order to accomplish a job. For instance, in the manufacturing environment, a drilling job could have activities such as: drilling-machine set-up, loading of a drilling job on a drilling-machine, actual drilling operation, unloading of a drilling job from a drilling-machine, etc.

# References

[Aben, 1993]

Aben, M. Formally Specifying Re-usable Knowledge Model Components. *Knowledge Acquisition*, 5 (2), pp. 119-141. 1993.

[Allen and Lehrer, 1992]

Allen, J. and Lehrer, N. DARPA/Rome Laboratory Planning and Scheduling Initiative Knowledge Representation Specification Language (KRSL). *Reference Manual*, Version 2.0.1, ISX Corporation. 1992.

[Agin, 1966]

Agin, N. Optimum seeking with branch and bound. *Management Science*, 13, pp. 176-185. 1966.

[Allen, 1983]

Allen, J. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26 (11), pp. 832-843. 1983.

[Angele et al., 1998]

Angele, J., Fensel, D., Landes, D., and Studer, R. Developing knowledge-based system with MIKE. *Automated software engineering*, 5 (4), pp. 389-418. 1998.

[Arpirez-Vega et al., 1998]

Arpirez-Vega, J. C., Gomez-Perez, A., Lozano-Tello, A., and Pinto, H. S. (ONTO)<sup>2</sup> Agent: an ontology-based WWW broker to select ontologies. In A. Gomez-Perez and V. R. Benjamins (Eds.) *Proceedings of Workshop on Applications of Ontologies and Problem-Solving Methods of the 13<sup>th</sup> European Conference on Artificial Intelligence (ECAI'98)*, Brighton, UK. 1998.

[Bacchus et al., 2002]

Bacchus, F., Chen, X., Beek, P. V., and Walsh, T. Binary vs. non-binary constraints. *Artificial Intelligence*, 140 (1-2), pp. 1-37. 2002.

[Bagchi et al., 1991]

Bagchi, S., Uckun, S., Miyabe, Y., and Kawamura, K. Exploring problem-specific re-combination operators for job shop scheduling. In R. K. Belew and L. B. Booker (Eds.) *Proceedings of the 4<sup>th</sup> International Conference on Genetic Algorithms*, San Mateo, Morgan Kaufmann, pp. 10-17. 1991.

[Baker, 1974]

- Baker, K. R. *Introduction to sequencing and scheduling*. John Wiley and Sons. 1974.
- [Balder *et al.*, 1993]
- Balder, J., van Harmelen, F., and Aben, M. A. KADS/(ML)2 Model of a Scheduling Task. In J. Treur and Th. Wetter (Eds.) *Formal Specification of Complex Reasoning system, Workshop series*, Ellis Horwood, pp. 15-44. 1993.
- [Baptiste and Le Pape, 1995]
- Baptiste, P. and Le Pape, C. Disjunctive Constraints for Manufacturing Scheduling: Principles and Extensions. In *Proceedings of the 3<sup>rd</sup> International Conference on Computer Integrated Manufacturing*, Singapore. 1995.
- [Baptiste, Le Pape, and Nuijten, 1995]
- Baptiste, P., Le Pape, C., and Nuijten, W. Constraint-Based Optimization and Approximation for Job-Shop Scheduling. In *Proceedings of the AAAI-SIGMAN Workshop on Intelligent Manufacturing Systems, IJCAI-95*, Canada. 1995.
- [Barros *et al.*, 1996]
- Barros L. Nunes de, Valente, A., and Benjamins, V. R. Modelling planning tasks. In *Proceedings of the 3<sup>rd</sup> International Conference on Artificial Intelligence Planning Systems (AIPS'96)*, pp. 11-18. American Association of Artificial Intelligence (AAAI). 1996.
- [Bartak, 1999]
- Bartak, R. On the boundary of Planning and Scheduling: A study. In *Proceedings of the 18<sup>th</sup> Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG'99)*, pp. 28-39, Manchester, UK. 1999.
- [Beck *et al.*, 1998]
- Beck, J. C., Davenport, A. J., Davis, E. D., and Fox, M. S. The ODO Project: Toward a Unified Basis for Constraint-Directed Scheduling. *Journal of Scheduling*, 1 (2), pp. 89-125. 1998.
- [Beck and Fox, 1998]
- Beck, J. C. and Fox, M. S. A Generic Framework for Constraint-Directed Search and Scheduling. *AI Magazine*, 19 (4), pp. 101-130. 1998.
- [Benders, 1962]

- Benders, J. F. Partitioning procedures for solving mixed-variable mathematical programming problems. *Numerische Mathematik*, 4, pp. 238-252. 1962.
- [Benjamins, 1993]
- Benjamins, V. R. Problem Solving Methods for Diagnosis. *PhD Thesis*, University of Amsterdam, Amsterdam, The Netherlands. 1993.
- [Benjamins, 1995]
- Benjamins, V. R. Problem solving methods for diagnosis and their role in knowledge acquisition. *International Journal of Expert Systems: Research & Applications*, 2 (8), pp. 93-120, 1995.
- [Benjamins and Aben, 1997]
- Benjamins, V. R. and Aben, M. Structure-preserving KBS Development Through Reusable Libraries: A Case-study in Diagnosis. *International Journal of Human-Computer Studies*, 47 (2), pp. 259-288. 1997.
- [Bernaras *et al.*, 1996]
- Bernaras, A., Laresgooiti, I., Corers, J. Building and Reusing Ontologies for Electrical Network Applications. In *W. Wahlster (Ed.) Proceedings of the 12<sup>th</sup> European Conference on Artificial Intelligence (ECAI'96)*, Budapest, Hungary, pp. 298-302, John Wiley. 1996.
- [Blazquez *et al.*, 1998]
- Blazquez, M., Fernández-Lopez, M., Garcia-Pinar, J. M., and Gomez-Perez, A. Building Ontologies at the Knowledge Level using the Ontology Design Environment. In *Proceedings of the 11<sup>th</sup> Workshop on Knowledge Acquisition, Modeling and Management*, Voyager Inn, Banff, Alberta, Canada, 18<sup>th</sup>-23<sup>rd</sup>, April. 1998.
- [Blythe, 1999]
- Blythe, J. Decision-Theoretic Planning. *AI Magazine*, 20 (2), pp. 37-54. 1999.
- [Blythe and Gil, 1999]
- Blythe, J. and Gil, Y. Problem-Solving Method for Plan Evaluation and Critiquing. In *Proceedings of the 12<sup>th</sup> Banff Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW'99)*, Voyager Inn, Banff, Alberta, Canada, 16<sup>th</sup>-21<sup>st</sup>, October. 1999.
- [Borgo *et al.*, 1996]

- Borgo, S., Guarino, N., and Masolo, C. Stratified ontologies: the case of physical objects. In *Proceedings of the workshop on Ontological Engineering at the 12<sup>th</sup> European Conference on Artificial Intelligence (ECAI'96)*, Budapest, pp. 5-15. 1996.
- [Borst *et al.*, 1995]
- Borst, P., Akkermans, J., Pos, A., and Top, J. The PhysSys ontology for physical systems. In B. Bredeweg (Ed.) *Working Papers of the 9<sup>th</sup> International Workshop on Qualitative Reasoning (QR'95)*, pp. 11-21. University of Amsterdam. 1995.
- [Breuker and Wielinga, 1985]
- Breuker, J. A. and Wielinga, B. J. KADS: Structured knowledge acquisition for expert systems. In *Proceedings of the 5<sup>th</sup> International Workshop on Expert Systems and their Applications*, Avignon, France. 1985.
- [Breuker and van de Velde, 1994]
- Breuker, J. A. and van de Velde, W. CommonKADS Library for Expertise Modelling: Reusable problem solving components. Netherlands, IOS Press. 1994.
- [Brucker *et al.*, 1999]
- Brucker, P., Drexl, A., Mohring, R., Neumann, K., and Pesch, E. Resource-constrained project scheduling: notation, classification, models and methods. *European Journal of Operations Research*, 112 (1), pp. 3-41. 1999.
- [Brusoni *et al.*, 1996]
- Brusoni, V., Console, L., Lamma, E., Mello, P., Milano, M., and Terenziani, P. Resource-based vs. Task-based Approaches for Scheduling Problems. In *Proceedings of the 9<sup>th</sup> International Symposium on Methodologies for Intelligent Systems (ISMIS'96)*, Zakopane, Poland, June 9-13, pp. 325-334. 1996.
- [Burke and Prossor, 1994]
- Burke, P. and Prosser, P. A Distributed Asynchronous Scheduler. In M. Zweben and M. S. Fox (Eds.) *Intelligent Scheduling*, Chapter 11, pp. 309-339. Morgan Kauffman, San Francisco, California. 1994.
- [Burke and Smith, 2000]

- Burke, E. K. and Smith, A. J. Hybrid evolutionary techniques for the maintenance scheduling problem. *IEEE Power Engineering Society Transactions*, 15 (1), pp. 122-128. 2000.
- [Bylander and Chandrasekaran, 1988]
- Bylander, T. and Chandrasekaran, B. Generic tasks in knowledge-based reasoning: The right level of abstraction for knowledge acquisition. In B. Gaines and J. Boose (Eds.) *Knowledge Acquisition for Knowledge Based Systems*, 1, pp. 65-77. Academic Press, London, 1988.
- [Caseau and Laburthe, 1995]
- Caseau, Y. and Laburthe, F. Improving Branch and Bound for Job-Shop Scheduling with Constraint Propagation. In *Proceedings of the 8<sup>th</sup> Franco-Japanese Conference CCS'95*, France. 1995.
- [Cesta et al., 1999]
- Cesta, A., Oddi, A., and Smith, S. F. An Iterative Sampling Procedure for Resource Constrained Project Scheduling with Time Windows. In *Proceedings of the 16<sup>th</sup> International Joint Conference on Artificial Intelligence (IJCAI'99)*, pp. 1022-1029, Stockholm. 1999.
- [Chandrasekaran, 1986]
- Chandrasekaran, B. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert*, 1, pp. 23-30. 1986.
- [Chandrasekaran, 1990]
- Chandrasekaran, B. Design Problem Solving: A Task Analysis. *AI Magazine*, 11(4), pp. 59-71. 1990.
- [Chandrasekaran et al., 1992]
- Chandrasekaran, B., Johnson T. R., and Smith J. W. Task-Structure Analysis for Knowledge Modelling. *Communications of the ACM*, 35 (9), pp. 124-137. 1992.
- [Cheng and Smith, 1995]
- Cheng, C. and Smith, S. Applying Constraint Satisfaction Techniques to Job Shop Scheduling. *Technical Report*, CMU-RI-TR-95-03, Robotics Institute, Carnegie Mellon University. 1995.
- [Clancy, 1986]
- Clancey, W. J. Heuristic Classification. *Artificial Intelligence*, 27 (3), pp. 289-350. 1986.

[Clancey, 1992]

Clancey, W. J. Model construction operators. *Artificial Intelligence*, 53 (1), pp. 1-135. 1992.

[Coelho and Lapalme, 1996]

Coelho, E. and Lapalme, G. Describing Reusable Problem-Solving Methods with a Method Ontology. In B. R. Gaines and M. Musen (Eds.) *Proceedings of the 10<sup>th</sup> Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Alberta, Canada, pp. 3-1-3-20. 1996.

[Collinot et al., 1988]

Collinot A., Le Pape C., and Pinoteau, G. SONIA: a knowledge-based scheduling system. *Artificial Intelligence in Engineering*, 3 (2), pp. 86-94. 1988.

[Conway et al., 1967]

Conway, R. W., Maxwell, W. L., and Miller, L. W. *Theory of Scheduling*. Addison Wesley, Reading, MA. 1967.

[Corkill, 1991]

Corkill, D. Blackboard systems. *AI Expert*, 6 (9), pp. 40-47. 1991.

[Dantzig and Wolfe, 1960]

Dantzig, G. B. and Wolfe, P. Decomposition principle for linear programs. *Operations Research*, 8 (1), pp. 101-111. 1960.

[Dantzig, 1991]

Dantzig, G. B. Linear Programming. In J. Lenstra and R. Kan (Eds.) *History of Mathematical Programming: A collection of Reminiscences*. CWI, Amsterdam. 1991.

[Davis, 1985]

Davis, L. Job shop scheduling with genetic algorithm. In *Proceedings of the International conference on Genetic Algorithms and Their Applications*, Carnegie Mellon University, pp. 136-140. 1985.

[De Kleer, 1986]

De Cleer, J. An assumption-based truth maintenance system. *Artificial Intelligence*, 28 (2), pp. 127-162. 1986.

[De Werra, 1985]

Werra, D. de. Introduction to time tabling. *European Journal of Operations Research*, 19 (2), pp. 151-162. 1985.

[Decthter et al., 1990]

- Dechter, R., Dechter, A., and Pearl, J. Optimization in constraint network. R. Oliver and J. Smith (Eds.) *Influence Diagrams, Belief Nets, and Decision Analysis*, Chicester, UK, Wiley, pp. 411-425. 1990.
- [Dechter and Pearl, 1985]
- Dechter, R. and Pearl, J. The Anatomy of Easy Problems: A Constraint-Satisfaction Formulation. In A. K. Joshi (Ed.) *Proceedings of the 9<sup>th</sup> International Joint Conference on Artificial Intelligence (IJCAI'85)*, Los Angeles, CA, Morgan Kaufmann. 1985.
- [Dechter and Meiri, 1989]
- Dechter, R. and Meiri, I. Experimental Evaluation of Preprocessing Techniques in Constraint Satisfaction Problems. In *Proceedings of the 11<sup>th</sup> International Joint Conference on Artificial Intelligence (IJCAI'89)*, San Mateo, CA, Morgan-Kaufman, pp. 217-277. 1989.
- [Dhar and Ranganathan, 1990]
- Dhar, V. and Ranganathan, N. Integer Programming versus Expert Systems: An Experimental Comparison. *Communications of the ACM*, 33 (3), pp. 323-336. 1990.
- [Domingue *et al.*, 1993]
- Domingue, J., Motta, E., and Watt, S. The emerging VITAL workbench. In N. Aussenac, G. Boy, B. R. Gaines, M. Linster, J. -G. Ganascia, and Y. Kodratoff (Eds.) *Proceedings of the 7<sup>th</sup> European Workshop on Knowledge Acquisition for Knowledge-based systems*, Sept. 6-10, pp. 320-339. 1993.
- [Domingue, 1998]
- Domingue, J. Tadzebao and WebOnto: Discussing, Browsing, and Editing Ontologies on the Web. In B. R. Gaines and M. A. Musen (Eds.) *Proceedings of the 11<sup>th</sup> Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, 18<sup>th</sup>-23<sup>rd</sup> April. 1998.
- [Dorndorf *et al.*, 2000]
- Dorndorf, U., Pesch, E., and Phan-Huy, T. Constraint propagation techniques for the disjunctive scheduling problem. *Artificial Intelligence*, 122 (1-2), pp. 189-240. 2000.
- [Dorn and Slany, 1994]
- Dorn, J. and Slany, W. A Flow Shop Compatibility Constraints in a Steel Manufacturing Plant. In M. Zweben and M. S. Fox (Eds.) *Intelligent*

- Scheduling*, Chapter 22, pp. 629-654, Morgan Kauffman, San Francisco, California. 1994.
- [Drexler, 1992]
- Drexler, E. K. *Nanosystems: Molecular Machinery, Manufacturing, and Computation*. John Wiley and Sons, Inc. Palo Alto, California. 1992.
- [Duda *et al.*, 1976]
- Duda, R. O., Hart, P. E., and Nilsson, N. J. Subjective Bayesian Methods for Rule Based Inference Systems. *Technical Note 124*, Stanford Research Institute, Artificial Intelligence Center, Menlo Park, CA, January, 1976.
- [Duda *et al.*, 1979]
- Duda, R. O., Hart, P. E., Konolige, K., and Reboh, R. A computer-based consultant for mineral exploration. *Annual Report*, Artificial Intelligence Centre, SRI International, Menlo Park, California. 1979.
- [Elleby *et al.*, 1988]
- Elleby, P., Fargher, H. E., and Addis, T. R. Reactive constraint-based scheduling. *IEE Colloquium on Artificial Intelligence in Planning of Production Control*. London, UK. 1988.
- [Erol *et al.*, 1994]
- Erol, K., Hendler, J., and Nau, D. S. Semantics for Hierarchical Task-Network Planning. *Technical report CS-TR-3239*, University of Maryland at College Park, 1994.
- [Fadel *et al.*, 1994]
- Fadel, F. G., Fox, M. S., and Gruninger, M. A Generic Enterprise Resource Ontology. In *Proceedings of the 3<sup>rd</sup> IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Morgantown, West Virginia. 1994.
- [Farquhar *et al.*, 1997]
- Farquhar, A., Fikes, R., and Rice, J. The Ontolingua server: a tool for collaborative ontology construction. *International Journal of Human-Computer Studies*, 46 (6), pp. 707-728. 1997.
- [Fatikow and Mounassypov, 1997]
- Fatikow, S. and Mounassypov, R. Assembly sequence planning for manufacturing by microrobots. In *Proceedings of the IEEE International Symposium on Assembly and Task Planning (ISATP'97): Towards Flexible*

and *Agile Assembly and Manufacturing*, pp. 269-274, New York, NY, USA. 1997.

[Fensel and van Harmelen, 1994]

Fensel, D. and van Harmelen, F. A Comparison of Languages which Operationalize and Formalize KADS Models of Expertise. *The Knowledge Engineering Review*, 9 (2), pp. 105-146. 1994.

[Fensel and Benjamins, 1998]

Fensel, D. and Benjamins, V. R. Key Issues for Automated Problem-Solving Method Reuse. In *Proceedings of the 13<sup>th</sup> European Conference on Artificial Intelligence (ECAI'98)*, Brighton, UK. John Wiley & Sons, Ltd. 1998.

[Fensel and Straatman, 1998]

Fensel, D. and Straatman, R. The essence of problem-solving methods: making assumptions to gain efficiency. *International Journal of Human-Computer Studies*, 48 (2), pp. 181-215. 1998.

[Fensel and Motta, 2001]

Fensel, D. and Motta, E. Structured Development of Problem-Solving Methods. *IEEE Transaction on Knowledge and Data Engineering*, 13 (6), pp. 913-932. 2001.

[Fernandez *et al.*, 1997]

Fernandez, M., Gomez-Perez, A., and Juristo, N. METHONTOLOGY: From Ontological Art Towards Ontological Engineering. In *Proceedings of the symposium on Ontological Engineering of AAAI*. Stanford, California. 1997.

[Foo and Takefuji, 1988]

Foo, Y. and Takefuji, Y. Stochastic neural networks for solving job-shop scheduling: Architecture and simulations (Part 2). In *Proceedings of the IEEE International Conference on Neural Networks, IEEE TAB*. 1988.

[Fox, 1981a]

Fox, M. S. An Organisation View of Distributed Systems. *IEEE Transaction on Systems, Man, and Cybernetics*, 11, pp. 70-80. 1981a.

[Fox, 1983]

Fox, M. S. Constraint-directed search: a case study of job-shop scheduling. *Technical Report, CMU-RI-TR-83-22*, Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania. 1983.

[Fox *et al.*, 1983]

Fox, M. S., Allen, B. P., Smith, S., and Strohm, G. A. Future knowledge-based systems for factory scheduling. In *Proceedings of the 12<sup>th</sup> Annual Technical CAMI Conference*, Dallas, Texas. 1983.

[Fox and Kempf, 1985]

Fox, B. R. and Kempf, K. G. Complexity, uncertainty, and opportunistic scheduling. In *Proceedings of the IEEE Conference Artificial Intelligence Applications*, Computer Society, 10662 Los Vaqueros Circle, Los Alamitos, California, 90720, pp. 487-492. 1985

[Fox and Sadeh, 1990]

Fox, M. S. and Sadeh, N. Why Scheduling is Difficult? A CSP Perspective. In *Proceedings of the 9<sup>th</sup> European Conference on Artificial Intelligence (ECAI'90)*, Stockholm, Sweden, pp. 754-767. 1990.

[French, 1982]

French, S. *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-shop*. Chichester: Ellis Horwood. 1982.

[Freund, 1979]

Freund, J. E. *Modern Elementary Statistics*. Prentice-Hall of India Private Limited, New Delhi. 1979.

[Freuder, 1982]

Freuder, E. C. A Sufficient Condition for Backtrack-Free Search. *Journal of ACM*, 29 (1), pp. 24-32. 1982.

[Freeman-Benson, Maloney, and Borning, 1990]

Freeman-Benson, B. N., Maloney, J., and Borning, A. An Incremental Constraint Solver. *Communications of the ACM*, 33 (1), pp. 54-63. 1990.

[Fikes and Nilson, 1971]

Fikes, R. and Nilson, N. J. STRIPS: A New Approach to the Application of Theorem Proving and Problem Solving. *Artificial Intelligence*, 2 (3-4), pp. 189-208. 1971.

[Fikes and Zhou, 2002]

Fikes, R. and Zhou, Q. A Reusable Time Ontology. In *Proceedings of the AAAI-2002 Workshop on Ontologies and the Semantic Web*, Edmonton, Canada. 2002.

[Gaines and Shaw, 1993]

- Gaines, B. R. and Shaw, M. L. G. Eliciting Knowledge and Transferring It Effectively to a Knowledge-Based System. *IEEE Transactions on Knowledge and Data Engineering*, 5 (1), pp. 4-14. 1993.
- [Garey and Johnson, 1979]
- Garey, M. R. and Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York. 1979.
- [Gaschnig, 1979]
- Gaschnig, J. A problem similarity approach to devising heuristics: first results. In *Proceedings of the 6<sup>th</sup> International Joint Conference on Artificial Intelligence (IJCAI'79)*, Tokyo, pp. 301-307. 1979.
- [Gaschnig, 1993]
- Gaschnig, M. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1, pp. 25-46. 1993.
- [Genesereth and Nilsson, 1987]
- Genesereth, M. R. and Nilsson, N. J. *Logical Foundation of Artificial Intelligence*. Los Altos, CA, Morgan Kaufmann. 1987.
- [Gennari *et al.*, 1994]
- Gennari, J., Tu, S., Rothenfluh, T., and Musen, M. Mapping domains to methods in support of reuse. In *B. R. Gaines and M. A. Musen (Eds.) Proceedings of the 8<sup>th</sup> Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, pp. 24:1-24:20, Banff, Canada. 1994.
- [Glover, 1986]
- Glover, F. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13 (5), pp. 533-549. 1986.
- [Glover, 1989]
- Glover, F. Tabu Search - Part I. *ORSA, Journal on Computing*, 1, pp. 190-206. 1989.
- [Glover, 1990]
- Glover, F. Tabu search - Part II. *ORSA, Journal of Computing*, 2, pp. 4-32. 1990.
- [Goldberg and Lingle, 1985]
- Goldberg, D. and Lingle, R. Alleles, loci, and the travelling salesman problem. *International Conference on Genetic Algorithms and Their Applications*, Carnegie Mellon University. 1985.

[Goldberg, 1989]

Goldberg, D. E. *Genetic Algorithms in Search Optimization and Machine Learning*. Addison-Wesley. 1989.

[Gomez-Perez and Benjamins, 1999]

Gomez-Perez, A. and Benjamins, V. R. Overview of Knowledge Sharing and Reuse: Ontologies and Problem Solving Methods. *Workshop on Ontologies and Problem-Solving Methods: Lessons Learned and Future Trends, International Joint Conference on Artificial Intelligence (IJCAI'99)*, 2 de Agosto, Estocolmo. 1999.

[Grant, 1986]

Grant, T. J. Lessons for OR from AI: A Scheduling Case Study. *Journal of Operations Research Society*, 37 (1), pp. 41-57. 1986.

[Gruber, 1993]

Gruber, T. R. Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5 (2), pp. 199-220. 1993.

[Gruber, 1995]

Gruber, T. R. Towards principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, 43 (5-6), pp. 907-928. 1995.

[Gruber and Oslen, 1994]

Gruber, T. R. and Olsen, G. R. An ontology for engineering mathematics. In Doyle, J., Torasso, P., and Sandewall, E. (Eds.) *Proceedings of the 4<sup>th</sup> International Conference on Principles of Knowledge Representation and Reasoning*, pp. 258-269, San Mateo, CA. Morgan Kaufmann. 1994.

[Grüninger and Fox, 1994]

Grüninger, M. and Fox, M. S. The Role of Competency Question in Enterprise Engineering. In *Proceedings of the IFIP WG 5.7 Workshop on Benchmarking: Theory and Practice*, Trondheim, Norway. 1994.

[Guarino, 1997]

Guarino, N. Understanding, building and using ontologies. *International Journal of Human-Computer Studies*, 46 (2-3), pp. 293-310. 1997.

[Guarino and Giaretta, 1995]

Guarino, N. and Giaretta, P. Ontologies and knowledge bases: Towards a terminological clarification. In N. Mars (Ed.) *Towards Very Large*

- Knowledge Bases: Knowledge Building and Knowledge Sharing*, pp. 25-32, IOS Press: Amsterdam, The Netherlands. 1995.
- [Gudes *et al.*, 1990]  
Gudes, E., Kuflik, T., and Meisels, A. On resource allocation by an expert system. *Engineering Applications of Artificial Intelligence*, 3 (2), pp. 101-109. 1990.
- [Guha and Lenat, 1990]  
Guha, R. V. and Lenat, D. B. Cyc: A midterm report. *AI Magazine*, 11 (3), pp. 32-59. 1990.
- [Hadavi *et al.*, 1992]  
Hadavi, K., Hsu, W-L., Chen, T., Lee, C-N. An Architecture for Real-Time Distributed Scheduling. *AI Magazine*, 13 (3), pp. 46-56. 1992.
- [Hama *et al.*, 1992a]  
Hama, T., Hori, M., and Nakamura, Y. Modelling Job Assignment Problems Based on Task Ontology. In *Proceedings of the 2<sup>nd</sup> Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop (JKAW'92)*, pp. 199-213, Kobe and Hatoyama, Japan. 1992.
- [Hama *et al.*, 1992b]  
Hama, T., Hori, M., and Nakamura, Y. Task-Specific Language Constructs for Describing Constraints in Job Assignment Problems. *Research Report, RT0084*, IBM. 1992.
- [Haralick and Elliot, 1980]  
Haralick, R. M. and Elliott, G. L. Increasing Tree Search Efficiency for Constraint Satisfaction Problems. *Artificial Intelligence*, 14 (3), pp. 263-313. 1980.
- [Hillier and Libermann, 1974]  
Hillier, F. S. and Libermann, G. *Introduction to Operations Research*. Holden-Day, San Francisco, USA. 1974.
- [Hopfield and Tank, 1985]  
Hopfield, J. and Tank, D. Neural computation of decisions in optimisation problems. *Biological Cybernetics*, 52, pp. 141-152. 1985.
- [Hori *et al.*, 1994]  
Hori, M., Nakamura, Y., and Hama, T. Configuring problem-solving methods: a CAKE perspective. *Knowledge Acquisition*, 6 (4), pp. 461-488. 1994.

[Hori *et al.*, 1995]

Hori, M., Nakamura, Y., Satoh, H., Maruyama, K., Hama, T., Honda, S., Takenaka, T., and Sekine, F. Knowledge-level analysis for eliciting composable scheduling knowledge. *Artificial Intelligence in Engineering*, 9 (4), pp. 253-264. 1995.

[Hori and Yoshida, 1998]

Hori, M. and Yoshida, T. Domain-oriented library of scheduling methods: design principles and real-life applications. *International Journal of Human-Computer Studies*, 49 (4), pp. 601-626. 1998.

[Ikeda *et al.*, 1998]

Ikeda, M., Seta, K., Kakusho, O., and Mizoguchi, R. An Ontology for building conceptual Problem Solving Model. In *Proceedings of the 13<sup>th</sup> European Conference on Artificial Intelligence (ECAI'98)*, Brighton, England. 1998.

[Jackson, 1956]

Jackson, J. R. An Extension of Johnson's Result on Job Lot Scheduling. *Naval Research Logistics Quarterly*, 3 (3), pp. 201-203. 1956.

[Jain and Meeran, 1998]

Jain, A. S. and Meeran, S. A state-of-the-art review of job-shop scheduling techniques. *Technical Report*, Department of Applied Physics, Electronic and Mechanical Engineering, University of Dundee, Dundee. 1998.

[Jeffcoat and Bulfin, 1993]

Jeffcoat, D. and Bulfin, R. Simulated annealing for resource-constrained scheduling. *European Journal of Operational Research*, 70, pp. 43-51. 1983.

[Jennings & Woolridge, 1998]

Jennings, N. R. and Woolridge, M. *Applications of intelligent agents. Agent Technology: Foundations, Applications, and Markets*. N. R. Jennings, M. Woolridge (Eds.), 1998.

[Jo *et al.*, 1997]

Jo, Geun-Sik, Jung, Jong-Jin, and Yang, Chang-Yoon. Expert Systems for Scheduling in an Airline Gate Allocation. *Expert Systems with Applications*, 13 (4), pp. 275-282. 1997.

[Johnson, 1954]

- Johnson, S. M. Optimal two and three-stage production schedules with set-up times included. *Naval Research Logistics Quarterly*, 1, pp. 61-68. 1954.
- [Jones and Rabelo, 1998]
- Jones, A. and Rabelo, J. C. Survey of Job Shop Scheduling Techniques. Gaithersburg, MD, NISTIR, National Institute of Standards and Technology. 1998.
- [Kandikjan and Dukovski, 1995]
- Kandikjan, T. and Dukovski, V. Automated planning and evaluation of product assembly sequences. In *Proceedings of the 31<sup>st</sup> International METADOR Conference*, San Mateo, CA, USA, pp. 541-548. 1995.
- [Kempf *et al.*, 1991]
- Kempf, K., Le Pape, C., Smith, S. F., and Fox, B. R. Issues in the Design of AI-Based Schedulers: A Workshop Report. *AI Magazine*, 11 (5), pp. 37-46. 1991.
- [Kirkpatrick *et al.*, 1983]
- Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. Optimization by simulated annealing. *Science*, 220 (4598), pp. 671-680. 1983.
- [Klee and Minty, 1972]
- Klee, V. and Minty, G. J. How good is the simplex algorithm? In *O. Shisha (Ed.) Inequalities, III*, Academic Press, New York, NY, pp. 159-175. 1972.
- [Krucky, 1994]
- Krucky, J. Fuzzy family setup assignment and machine balancing. *Hewlett-Packard Journal*, 45 (3), pp. 51-64. 1994.
- [Kruger, 1992]
- Kruger, C. Software reuse. *Computing Surveys*, 24 (2), pp. 131-183. 1992.
- [Kumar, 1992]
- Kumar, V. Algorithms for Constraint Satisfaction Problems: A Survey. *AI Magazine*, 13 (1), pp. 32-44. 1992.
- [Lawler and Wood, 1966]
- Lawler, E. and Wood, D. Branch and bound methods: a survey. *Operations Research*, 14, pp. 699-719. 1966.
- [Lawler, 1983]

- Lawler, E. L. (Ed.). *Recent results in the theory of machine scheduling. Mathematical Programming: the state-of-the-art*. Springer-Verlag. 1983.
- [Lee *et al.*, 1995]
- Lee, J. K., Lee, K. J., Hong, J. S., Kim, W., Kim, E. Y., Choi, S. Y., Kim, H. D., Yang, O. R., and Choi, H. R. DAS: Intelligent Scheduling Systems for Shipbuilding. *AI Magazine*, 16 (2), pp. 78-94. 1995.
- [Lessila *et al.*, 1996]
- Lassila, O., Becker, M., and Smith, S. An exploratory prototype for aero-medical evacuation planning. *Technical Report, CMU-RI-TR-96-02*, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA. 1996.
- [Le Pape and Sauvé, 1985]
- Le Pape, C. and Sauvé, B. SOJA: an expert system for workshop daily planning. In *Proceedings of the Avignon-85*, Avignon, France, pp. 849-867. 1985.
- [Le Pape, 1994]
- Le Pape, C. Implementation of Resource Constraints in ILOG SCHEDULE: A library for the development of Constraint-Based Scheduling Systems. *Intelligent Systems Engineering*, 3 (2), pp. 55-66. 1994.
- [Le Pape, 1995]
- Le Pape, C. Three Mechanisms for Managing Resource Constraints in a Library for Constraint-Based Scheduling. In *Proceedings of the INRIA/IEEE Conference on Emerging Technologies and Factory Automation*, Paris, France. 1995.
- [Lesser and Corkill, 1981]
- Lesser, V. and Corkill, D. Functionally Accurate, Cooperative Distributive Systems. *IEEE Transaction on Man, Systems, and Cybernetics*, 11 (1), pp. 81-98. 1981.
- [Liebowitz and Potter, 1995]
- Liebowitz, J. and Potter, W. E. Scheduling Objectives, Requirements, Resources, Constraints, and Processes: Implications for a Generic Expert Scheduling System Architecture and Toolkit. *Expert Systems with Applications*, 9 (3), pp. 423-432. 1995.
- [Liu, 1988]

- Liu, B. Scheduling via reinforcement. *Artificial Intelligence in Engineering*, 3 (2), pp. 76-85. 1988.
- [Lloyd, 1982]
- Llyod, E. L. Critical path scheduling with resource and processor constraints. *Journal of ACM*, 29 (3), pp. 781-811. 1982.
- [Lo and Bavarian, 1991]
- Lo, Z. and Bavarian, B. Scheduling with neural networks for flexible manufacturing systems. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Sacramento, California. 1991.
- [Macworth, 1977]
- Macworth, A. K. Consistency in Networks of Relations. *Artificial Intelligence*, 8, pp. 99-118. 1977.
- [Marcus, 1988]
- Marcus, S. *Automating Knowledge Acquisition for Expert Systems*. Kluwer Academic Publishers. 1988.
- [Marcus *et al.*, 1988]
- Marcus, S., Stout, J., and McDermott, J. VT: an expert elevator designer that uses knowledge-based backtracking. *AI Magazine*, 9 (1), pp. 95-111. 1988.
- [Marcus and McDermott, 1989]
- Marcus, S. and McDermott, J. SALT: A Knowledge Acquisition Language for Propose-and-Revise Systems. *Artificial Intelligence*, 39 (1), pp. 1-37. 1989.
- [Maxwell and Howell, 1995]
- Maxwell, T. and Howell, E. Planning as a Precursor to Scheduling for Space Station Payload Operations. In *Proceedings of the American Institute of Aeronautics and Astronautics (AIAA'95) Space Programs and Technologies Conference*, Huntsville, Alabama, USA. 1995.
- [McDermott, 1988]
- McDermott, J. Towards a taxonomy of problem solving methods. In S. Marcus (Ed.) *Automating Knowledge Acquisition for Expert Systems*, Kluwer Academic. 1988.
- [McGuinness and Harmelen, 2004]

- McGuinness, D. L. and Harmelen, F. van. OWL Web Ontology Language Overview. *Editors, W3C Recommendation*, 10 February, 2004, <http://www.w3c.org/TR/2004/REC-owl-guide-20040210/>.
- [McKay, Safayani, and Buzacott, 1988]
- McKay, K. N., Safayani, F. R., and Buzacott, J. A. Job-Shop Scheduling Theory: What is Relevant? *Interfaces*, 18 (4), pp. 84-90. 1988.
- [McKenzie, 1976]
- McKenzie, L. Turnpike theory. *Econometrics*, 44 (5), pp. 841-865. 1976.
- [Meng and Sullivan, 1991]
- Meng, C-C. and Sullivan, M. LOGOS: A Constraint-Directed Reasoning Shell for Operations Management. *IEEE Expert*, 6 (1), pp. 20-28. 1991.
- [Merkle, 1997]
- Merkle, R. C. Convergent assembly. *Nanotechnology*, 8 (1), pp. 18-22. 1997.
- [Merkle, 1996]
- Merkle, R. C. Design considerations for an assembler. *Nanotechnology*, 7, pp. 210-215. 1996.
- [Minton *et al.*, 1992]
- Minton, S., Johnston, M., Philips, A., and Laird, P. Minimizing Conflicts: A Heuristic Repair Method for Constraint-Satisfaction and Scheduling Problems. *Artificial Intelligence*, 58 (1-3), pp. 161-205. 1992.
- [Mittal and Frayman, 1989]
- Mittal, S. and Frayman, F. Towards a generic model of configuration tasks. In *Proceedings of the 11<sup>th</sup> International Joint Conference on Artificial Intelligence (IJCAI'89)*, pp. 1395-1401, San Mateo, CA, Morgan Kaufman. 1989.
- [Miyashita and Sycara, 1994]
- Miyashita, K. and Sycara, K. Adaptive case-based control of schedule revision. In M. Zweben and M. S. Fox (Eds.) *Intelligent Scheduling*, Chapter 10, pp. 291-308, Morgan Kauffman, San Francisco, California. 1994.
- [Mizoguchi *et al.*, 1995]
- Mizoguchi, R., Vanwelkenhuysen, J., and Ikeda, M. Task Ontology for Reuse of Problem Solving Knowledge. *Towards Very Large Knowledge Bases*, pp. 46-57. IOS Press. 1995.

[Mott *et al.*, 1988]

Mott, D. H., Cunningham, J., Kelleher, G., Gadsden, J. A. Constraint-Based Reasoning for Generating Naval Flying Programmes. *Expert systems*, 3 (2), pp. 226-246. 1988.

[Motta and Zdrahal, 1996]

Motta, E. and Zdrahal, Z. Parametric Design Problem Solving. In B. R. Gaines and M. Musen (Eds.) *Proceedings of the 10<sup>th</sup> Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada. 1996.

[Motta and Zdrahal, 1998]

Motta, E. and Zdrahal, Z. A library of problem-solving components based on the integration of the search paradigm with task and method ontologies. *International Journal of Human-Computer Studies*, 49 (4), pp. 437-470. 1998.

[Motta, 1999]

Motta, E. *Reusable Components for Knowledge Modelling - Principles and Case Studies in Parametric Design Problem Solving*. IOS Press, Amsterdam, The Netherlands. 1999.

[Motta, 2001]

Motta, E. The knowledge modeling paradigm in knowledge engineering. In S.K. Chang (Ed.) *Handbook of Software Engineering and Knowledge Engineering*, World Scientific Pub. Co. ISBN: 981-02-4973-X. 2001.

[Motta *et al.*, 2003]

Motta, E., Domingue, J., Cabral, L., and Gaspari, M. IRS-II: A Framework and Infrastructure for Semantic Web Services. In *Proceedings of the 2<sup>nd</sup> International Semantic Web Conference (ISWC'03)*, 20<sup>th</sup>-23<sup>rd</sup> October, Sundial Resort, Sanibel Island, Florida. USA. 2003.

[Muscettola *et al.*, 1992]

Muscettola, N., Smith, S., Cesta, A., and D'Aloisi, D. Coordinating Space Telescope Operations in an Integrated Planning and Scheduling Framework. *IEEE Control Systems*, 12 (1), pp. 28-37. 1992.

[Musen, 1992]

Musen, M. Overcoming the limitations of role-limiting methods. *Knowledge Acquisition*, 4 (2), pp. 165-170. 1992.

[Musen *et al.*, 1993]

- Musen, M., Tu, S. W., Eriksson, H., Gennari, J. H., and Puerta, A. R. PROTÉGÉ-II: An Environment for Reusable Problem-Solving Methods and Domain Ontologies. In *Proceedings of the 13<sup>th</sup> International Joint Conference on Artificial Intelligence (IJCAI'93)*, Chambéry, Savoie, France. 1993.
- [Musen *et al.*, 1994]
- Musen, M., Gennari, J., Eriksson, H., Tu, S., and Puerta, A. PROTÉGÉ-II: Computer Support for Development of Intelligent Systems from Libraries of Components. *Technical Report, KSL-94-60*, Stanford University, Knowledge Systems Laboratory. 1994.
- [Nau *et al.*, 1999]
- Nau, D. S., Cao, Y., Lotem, A., Munoz-Avila, H. SHOP: Simple Hierarchical Ordered Planner. In *Proceedings of the 6<sup>th</sup> International Joint Conference on Artificial Intelligence, (IJCAI'99)*, Stockholm, Sweden, pp. 968-975. 1999.
- [Navinchandra and Marks, 1987]
- Navinchandra, D. and Marks, D. H. Layout Planning as Constraint Labelling Optimization Problem. In *Proceedings of the 4<sup>th</sup> International Symposium on Robotics and AI in Construction*, Haifa, Israel. 1987.
- [Neches *et al.*, 1991]
- Neches, R., Fikes, R. E., Finin, T., Gruber, T. R., Senator, T., and Swartout, W. R. Enabling technology for knowledge sharing. *AI Magazine*, 12 (3), pp. 36-56. 1991.
- [Newell and Simon, 1976]
- Newell, A. and Simon, H. A. Computer science as empirical enquiry: Symbols and Search. *Communications of the ACM*, 19 (3), pp. 113-126. 1976.
- [Newell, 1982]
- Newell, A. The Knowledge Level. *Artificial Intelligence*, 18 (1), pp. 87-127. 1982.
- [Nickols, 2000]
- Nickols, F. The knowledge in knowledge management. In J. W. Cortada and J. A. Woods (Eds.) *The knowledge management yearbook 2000-2001*, pp. 12-22. Butterworth-Heinemann. 2000.
- [Nii, 1986]

- Nii, H. P. Blackboard systems: the blackboard model of problem solving and the evolution of blackboard architecture. *AI Magazine*, 7 (2), pp. 38-53. 1986.
- [Nonaka and Takeuchi, 1995]  
Nonaka, I. and Takeuchi, H. *The knowledge creating company*. Oxford University Press, Oxford, New York. 1995.
- [Noronha and Sarma, 1991]  
Noronha, S. J. and Sarma, V. V. S. Knowledge-Based Approaches for Scheduling Problems: A Survey. *IEEE Transaction on Knowledge and Data Engineering*, 3 (2), pp. 160-171. 1991.
- [Nowicki and Smutnicki, 1996]  
Nowicki, E. and Smutnicki, C. A fast taboo search algorithm for the job shop problem. *Management Science*, 42 (6), pp. 797-813. 1996.
- [Ow et al., 1988]  
Ow, P. S., Smith, S. F., and Thiriez, A. Reactive plan revision. In *Proceedings of the 7<sup>th</sup> National Conference on Artificial Intelligence*, pp. 77-82, Menlo Park, California. AAAI Press. 1988.
- [Panwalkar and Iskander, 1977]  
Panwalkar, S. S. and Iskander, W. A Survey of Scheduling Rules. *Operations Research*, 25 (1), pp. 45-61. 1977.
- [Parunak et al., 1985]  
Parunak, H., Irish, B., Kindrick, J., and Lozo, P. Fractal actors for distributed manufacturing control. In *Proceedings of the 2<sup>nd</sup> IEEE Conference on Artificial Intelligence Applications*, Washington: IEEE Computer Society, pp. 653-660. 1985.
- [Petrie et al., 1989]  
Petrie, C., Causey, R., Steiner, D., and Dhar, V. A Planning Problem: Revisable Academic Course Scheduling. *Technical Report, AI-020-89*, MCC Corp., Austin, Texas. 1989.
- [Pinedo, 2001]  
Pinedo, M. *Scheduling. Theory, Algorithms and Systems*. 3<sup>rd</sup> Edition, Prentice Hall. 2001.
- [Poeck and Puppe, 1992]  
Poeck, K. and Puppe, F. COKE: Efficient solving of complex assignment problems with the propose-and-exchange method. In *Proceedings of the*

- 5<sup>th</sup> International Conference on Tools with Artificial Intelligence, pp. 136-143, Arlington, Virginia, USA. 1992.
- [Poeck and Gappa, 1993]
- Poeck, K. and Gappa, U. Making Role-Limiting Shells More Flexible. In *Proceedings of the 7<sup>th</sup> Workshop on Knowledge Acquisition for Knowledge-Based Systems*, pp. 103-122, Toulouse and Caylus. 1993.
- [Poli, 2002]
- Poli, R. Ontological Methodology. *International Journal of Human-Computer Studies*, 56 (6), pp. 639-664. 2002.
- [Preece et al., 1996]
- Preece, A. D., Groosner, C., and Radhakrishnan, T. Validating dynamic properties of rule-based systems. *International Journal of Human-Computer Studies*, 44 (2), pp. 145-169. 1996.
- [Prosser, 1989]
- Prosser, P. A. Reactive Scheduling Agent. In *Proceedings of the 11<sup>th</sup> International Joint Conference on Artificial Intelligence, (IJCAI'89)*, Detroit, USA, pp. 1004-1009. 1989.
- [Prosser and Buchanan, 1994]
- Prosser, P. and Buchanan, I. Intelligent scheduling: past, present, and future. *Intelligent Systems Engineering*, 3 (2), pp. 67-78. 1994.
- [Puerta et al., 1992]
- Puerta, A. R., Egar, J., Tu, S., and Musen, M. A. Multiple-method shell for the automatic generation of knowledge acquisition tools. *Knowledge Acquisition*, 4 (2), pp. 171-196. 1992.
- [Rescher and Urquhart, 1971]
- Rescher, N. and Urquhart, A. *Temporal Logic*. New York, Springer-Verlag. 1971.
- [Rabelo, 1990]
- Rabelo, L. *Hybrid Artificial Neural Networks and Knowledge-Based Expert Systems Approach to Flexible Manufacturing Systems Scheduling*. University of Missouri-Rolla. 1990.
- [Ravidran, Philips, and Solberg, 1987]
- Ravidran, A., Philips, D. T., and Solberg, J. J. *Operations Research Principles and Practice*. John Wiley & Sons Inc. 1987.
- [Render and Stair, 1982]

- Render, B. and Stair, R. M. *Qualitative Analysis for Management*. Allyn & Bacon Inc., Massachusetts. 1982.
- [Ricci, 1990]
- Ricci, F. Equilibrium Theory and Constraint Networks. *Technical Report, 9007-08*, Istituto per la Ricerca Scientifica e Tecnologica, Povo, Italy. 1990.
- [Rumelhart *et al.*, 1986]
- Rumelhart, D. and McClelland, J. *Parallel Distributed Processing: Explorations in Microstructure of Cognition*. MIT Press, Cambridge, Massachusetts. 1986.
- [Runkel and Birmingham, 1993]
- Runkel, J. T. and Birmingham, W. B. Knowledge acquisition in the small: building knowledge acquisition tools from pieces. *Knowledge Acquisition*, 5 (2), pp. 221-243. 1993.
- [Runkel *et al.*, 1994]
- Runkel, J. T., Birmingham, W. B., and Balkany, A. Separation of Knowledge: A Key to Reusability. In B. R. Gaines and M. Musen (Eds.) *Proceedings of the 8<sup>th</sup> Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, 2 (3), pp. 36:1-36:19, Banff, Canada. SRDG Publications. 1994.
- [Runkel *et al.*, 1996]
- Runkel, J. T., Birmingham, W. B., and Balkany, A. Solving VT by Reuse. *International Journal of Human-Computer Studies*, 44 (3-4), pp. 403-433. 1996.
- [Sadeh and Fox, 1996]
- Sadeh, N. and Fox, M. S. Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem. *Artificial Intelligence*, 86 (1), pp. 1-41. 1996.
- [Sadeh, 1991]
- Sadeh, N. M. Lookahead Techniques for Micro-opportunistic Job-Shop Scheduling. *PhD Thesis, CMU-CS-91-102*, Computer Science Department, Carnegie-Mellon University, Carnegie-Mellon. 1991.
- [Sadeh, 1994]
- Sadeh, N. Micro-Opportunistic Scheduling: The Micro-Boss Factory Scheduler. In M. Zweben and M. S. Fox (Eds.) *Intelligent Scheduling*,

- Chapter 4, pp. 99-135, Morgan Kaufman, San Francisco, California. 1994.
- [Sathi *et al.*, 1985]  
Sathi, A., Fox, M. S., and Greenberg, M. Representation of Activity Knowledge for Project Management. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 7 (5), pp. 531-552. 1985.
- [Saucer, 1997]  
Saucer, J. Knowledge-Based Systems Techniques and Applications in Scheduling. In T. L. Leondes (Ed.) *Knowledge-Based Systems Techniques and Applications*, 1 (4). Academic Press, San Diego. 1997.
- [Saucer, 2001]  
Saucer, J. Knowledge-Based Design of Scheduling Systems. An *International Journal of Intelligent Automation and Soft Computing*, 7 (1), pp. 55-62. 2001.
- [Schreiber *et al.*, 1994]  
Schreiber, G., Wielinga, B., Hogg, R. de, Akkermans, H., and Van de Velde, W. CommonKADS: A Comprehensive Methodology for KBS Development. *IEEE Expert*, 9 (6), pp. 28-37. 1994.
- [Sharma, 1998]  
Sharma, S. D. *Operations Research*. Kedarnath – Ramnath & Co., Meerut, India. 1996.
- [Shapiro, 1979]  
Shapiro, J. F. A survey of lagrangian techniques for discrete optimization. *Annals of Discrete Mathematics*, 5, pp. 113-138. 1979.
- [Slagle *et al.*, 1984]  
Slagle, J. R., Gaynor, M. W., and Halpern, E. H. An intelligent control strategy for computer consultation. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 6 (2), pp. 129-136. 1984.
- [Slagle and Hamburger, 1985]  
Slagle, J. R. and Hamburger, H. An expert system for a resource allocation problem. *Communications of ACM*, 28 (9), pp. 994-1004. 1985.
- [Slany, 1994]  
Slany, W. Fuzzy scheduling. *PhD Thesis*, Technical University of Vienna, Vienna. 1994.
- [Smith, 1980]

- Smith, R. G. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers*, 29 (12), pp. 1104-1113. 1980.
- [Smith *et al.*, 1990]
- Smith, S., Ow, P. S., Muscettola, N., Potvin, J. Y., and Matthys, D. OPIS: An Opportunistic Factory Scheduling System. In *Proceedings of the 3<sup>rd</sup> International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, vol. 1, 15<sup>th</sup>-18<sup>th</sup> July, pp. 268-274. 1990.
- [Smith, 1994]
- Smith, S. F. OPIS: A Methodology and Architecture for Reactive Scheduling. In M. Zweben and M. S. Fox (Eds.) *Intelligent Scheduling*, Chapter 2, pp. 29-66. Morgan Kauffman, San Francisco, California. 1994.
- [Smith and Becker, 1997]
- Smith, S. and Becker, M. A. An Ontology for Constructing Scheduling Systems. In working Notes of *AAAI Symposium on Ontological Engineering*, Stanford, CA. 1997.
- [Smith and Goodwin, 1995]
- Smith, D. and Goodwin, S. D. Constraint-Based Intelligent Scheduling. *Technical report, CS-95-02*, University of Regina, Regina, Saskatchewan, S4S 0A2. 1995.
- [Smith *et al.*, 2000]
- Smith, D., Frank, J., and Jonsson, A. K. Bridging the gap between planning and scheduling. *The Knowledge Engineering Review*, 15 (1), pp. 47-83. 2000.
- [Starkweather *et al.*, 1993]
- Strakweather, T., Whitley, D., and Cookson, B. A Genetic Algorithm for Scheduling with Resource Consumption. In *Proceedings of the Joint German/US Conference on Operations Research in Production Planning and Control*, pp. 567-583. 1993.
- [Stallman and Sussman, 1977]
- Stallman, R. M. and Sussman, G. J. Forward Reasoning and Dependency Directed Backtracking in a System for Computer-Aided Circuit Analysis. *Artificial Intelligence*, 9 (2), pp. 135-196. 1977.
- [Steels, 1990]

- Steels, L. Components of Expertise. *AI Magazine*, 11 (2), pp. 30-49. 1990.
- [Stout *et al.*, 1988]
- Stout, J., Caplain, G., Marcus, S., and McDermott, J. Towards automating recognition of differing problem-solving demands. *International Journal of Man Machine Studies*, 29 (5), pp. 599-611. 1988.
- [Sundin, 1994]
- Sundin, U. Assignment and Scheduling. In J. Breuker and W. van de Velde (Eds.) *CommonKADS Library for Expertise Modelling*, pp. 231-263. IOS Press, Amsterdam, The Netherlands. 1994.
- [Swartout and Gil, 1995]
- Swartout, B. and Gil, Y. EXPECT: Explicit Representations for Flexible Acquisition. In B. R. Gaines and M. A. Musen (Eds.) *Proceedings of the 9<sup>th</sup> Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Alberta, Canada. 1995.
- [Swartout *et al.*, 1997]
- Swartout, B., Ramesh, P., Knight, K., and Russ, T. Toward Distributed Use of Large-Scale Ontologies. In *AAAI-97 Spring Symposium Series on Ontological Engineering*, Stanford, California. 1997.
- [Tate, 1994]
- Austin Tate. Plan Ontology - A Working Document. In *Proceedings of the Workshop on Ontology Development and Use*, Le Jolle, CA. 1994.
- [Tsang, 1993]
- Tsang, E. P. K. *Foundations of constraint satisfaction*. Academic Press. 1993.
- [Talbot, 1982]
- Talbot, F. B. Resource-constrained project scheduling with time-resource tradeoff: the nonpreemptive case. *Management Science*, 28 (10), pp. 1197-1210. 1982.
- [ten Teije, 1997]
- ten Teije, A. Automated Configuration of Problem Solving Methods in Diagnosis. *PhD Thesis*, University of Amsterdam, Amsterdam, The Netherlands. 1997.
- [Terpstra *et al.*, 1993]
- Terpstra, P., van Heijst, G., Wielinga, B., and Shadbolt, N. Knowledge acquisition support through generalised directive models. In J. M. David,

- J. P. Krivine, and R. Simmons (Eds.) *Second Generation Expert Systems*. Springer-Verlag, pp. 428-455. 1993.
- [Tijerino and Mizoguchi, 1993]
- Tijerino, Y. A. and Mizoguchi, R. MULTIS II: enabling end-users to design problem-solving engines via two-level task ontologies. In N. Aussenac-Gilles, G. A. Boy, B. R. Gaines, J. Ganascia, Y. Kodratoff, and M. Linster (Eds.) *7<sup>th</sup> European Workshop on Knowledge Acquisition for Knowledge-Based Systems (EKAW'93)*, Lecture Notes in Computer Science, Springer, pp. 340-359, Toulouse and Caylus, France. 1993.
- [Tsujimura *et al.*, 1993]
- Tsujimura, Y., Park, S., Chang, S., and Gen, M. An effective method for solving flow shop scheduling problems with fuzzy processing times. *Computers and Industrial Engineering*, 25 (1-4), pp. 239-242. 1993.
- [Tu *et al.*, 1995]
- Tu, S. W., Eriksson, H., Gennari, J., Shahar, Y., and Musen, M. A. Ontology-Based Configuration of Problem-Solving Methods and Generation of Knowledge-Acquisition Tools: Application of PROTÉGÉ-II to Protocol-Based Decision Support. *Artificial Intelligence in Medicine*, 7 (3), pp. 257-289. 1995.
- [Tu and Musen, 1996]
- Tu, S. W. and Musen, M. A. Episodic Refinement of Episodic Skeletal Plan Refinement. *Technical Report, SMI-96-0647*, Stanford, Stanford University School of Medicines. 1996.
- [Uckun *et al.*, 1993]
- Uckun, S., Bagchi, S., Kawamura, K., and Miyabe, Y. Managing genetic search in job shop scheduling. *IEEE Expert*, 8 (5), pp. 15-24. 1993.
- [Uschold and King, 1995]
- Uschold, M. and King, M. Towards a methodology for building ontologies. In *Proceedings of the workshop on Basic Ontological Issues in Knowledge Sharing, International Joint Conference on Artificial Intelligence (IJCAI'95)*, Motreal. 1995.
- [Vaessens, 1995]
- Vaessens, R. J. M. Generalized Job Shop Scheduling: Complexity and Local Search. *Ph.D. thesis*, Eindhoven University of Technology. 1995.
- [Vakharia and Chang, 1990]

- Vakharia, A. and Chang, Y. A simulated annealing approach to scheduling a manufacturing cell. *Naval Research Logistics*, 37, pp. 559-577. 1990.
- [Valente and Breuker, 1996]
- Valente, A. and Breuker, J. A. Towards Principled Core Ontologies. In B. R. Gaines and M. A. Musen (Eds.) *Proceedings of the 10<sup>th</sup> Banff Knowledge Acquisition for Knowledge-based Systems Workshop*, Banff, Alberta, Canada. 1996.
- [Valente *et al.*, 1998]
- Valente, A., Benjamins, V. R., and Nunes De Barros, L. A library of system-derived problem-solving methods for planning. *International Journal of Human-Computer Studies*, 48 (4), pp. 417-447. 1998.
- [Valente and Löckenhoff, 1993]
- Valente, A. and Löckenhoff, C. Organization as guidance: A library of assessment models. In *Proceedings of the 7<sup>th</sup> European Knowledge Acquisition Workshop (EKAW'93)*, pp. 243-262. 1993.
- [van Heijst *et al.*, 1992]
- van Heijst, G., Terpstra, P., Wielinga, B. J., Shadbolt, N. Using Generalised Directive Models in Knowledge Acquisition. In *Proceedings of the 6<sup>th</sup> European Knowledge Acquisition Workshop*, Heidelberg and Kaiserslautern, Germany, pp. 112-132. 1992.
- [van Heijst, 1995]
- van Heijst, G. The Role of Ontologies in Knowledge Engineering. *PhD thesis*, University of Amsterdam. 1995.
- [van Heijst *et al.*, 1997]
- van Heijst, G., Schreiber, A. Th., and Wielinga, B. J. Using explicit ontologies in KBS development. *International Journal of Human-Computer Studies*, 46 (2), pp. 183-292. 1997.
- [Watson *et al.*, 2003]
- Watson, Jean-Paul, Beck, J. C., Howe, A. E., and Whitley, L. D. Problem-difficulty for tabu search in job-shop scheduling. *Artificial Intelligence*, 143 (2), pp. 189-217. 2003.
- [Wielinga *et al.*, 1992]
- Wielinga, B. J., van de Velde, W., and Schreiber, A. T. The Common KADS framework for knowledge modelling. In *Proceedings of the 7<sup>th</sup>*

- Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, pp. 31:1-31:29. 1992.
- [Wielinga *et al.*, 1995]
- Wielinga, B., Akkermans, J. M., and Schreiber, A. Th. A Formal Analysis of Parametric Design Problem Solving. In B. R. Gaines and M. A. Musen (Eds.) *Proceedings of the 9<sup>th</sup> Banff Knowledge Acquisition Workshop for Knowledge-Based System Workshop (KAW'94)*, Banff, Canada. 1995.
- [Wielinga and Schreiber, 1997]
- Wielinga, B. and Schreiber, G. Configuration-Design Problem Solving. *IEEE Expert*, 12 (2), pp. 49-56. 1997.
- [Wu, 1987]
- Wu, D. *An Expert Systems Approach for the Control and Scheduling of Flexible Manufacturing Systems*. Pennsylvania State University, Pennsylvania. 1987.
- [Zhang and Zhang, 1995]
- Zhang, M. and Zhang, C. The consensus of uncertainties in distributed expert systems. In *Proceedings of the 1<sup>st</sup> International Conference on Multi-Agent Systems*. MIT Press, Massachusetts. 1995.
- [Zhou *et al.*, 1990]
- Zhou, D., Cherkassky, T., Baldwin, T., and Hong, D. Scaling neural networks for job shop scheduling. In *Proceedings of the International Conference on Neural Network*, San Diego, CA, pp. 889-894. 1990.
- [Zhou and Fikes, 2000]
- Zhou, Q. and Fikes, R. A Reusable Time Ontology. *Technical Report, KSL-00-01*, Knowledge Systems Laboratory, Stanford University. 2000.
- [Zdrahal and Motta, 1995]
- Zdrahal, Z. and Motta, E. An In-depth Analysis of Propose & Revise Problem Solving Method. In B. R. Gaines and M. A. Musen (Eds.) *Proceedings of the 9<sup>th</sup> Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, pp. 38:1-38:20. 1995.
- [Zweben *et al.*, 1992]
- Zweben, M., Davis, E., Daun, B., Drascher, E., Deale, M., and Eskey, M. Learning to improve constraint-based scheduling. *Artificial Intelligence*, 58 (1-3), pp. 271-296. 1992.
- [Zweben *et al.*, 1993]

Zweben, M., Daun, B., Davis, E., and Deale, M. Scheduling and Rescheduling with Iterative Repair. *IEEE Transactions on System, Man, and Cybernetics*, 23 (6), pp. 1588-1595. 1993.

[Zweben and Fox, 1994]

Zweben, M. and Fox, M. S. (Eds.). *Intelligent Scheduling*. Morgan Kaufmann, San Francisco, California, USA. 1994.

# Appendix 1

## A COMPLETE SPECIFICATION OF THE SCHEDULING TASK ONTOLOGY

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Base: 10; Package: OCML; -*-
;;;THE OPEN UNIVERSITY
;;;Author: Dnyanesh Rajpathak
(in-ontology scheduling1)

(def-class SCHEDULING-TASK (goal-specification-task) ?task
  "Scheduling task is defined as an assignment of time constrained jobs to time constrained
  resources within a given time frame, which indicates the total time-horizon of a schedule.
  An admissible schedule will have to satisfy all the constraints imposed on jobs or
  resources while maintaining the requirements. The output to the scheduling task is a legal
  schedule in accordance with the solution criteria such as, complete, admissible and
  feasible."
  ((has-input-role :value has-jobs
                   :value has-activities
                   :value has-resources
                   :value has-hard-constraints
                   :value has-requirements
                   :value has-schedule-time-range
                   :value has-preferences
                   :value has-cost-function
                   :value has-cost-algebra)
   (has-output-role :value has-schedule-model :max-cardinality 1)
   (has-schedule-model :type schedule-model :max-cardinality 1)
   (has-jobs :type list :min-cardinality 1)
   (has-activities :type list :min-cardinality 1)
   (has-resources :type list :min-cardinality 1)
   (has-hard-constraints :type hard-constraint)
   (has-requirements :type requirement)
   (has-schedule-time-range :type time-range :max-cardinality 1)
   (has-preferences :type preference :cardinality 1)
   (has-cost-function :type cost-function :max-cardinality 1)
   (has-cost-algebra :default-value '(+ - <) :cardinality 1)
   (has-precondition :documentation "Scheduling task must have a job and a
                                   resource in order to generate schedule."
                     :value (kappa (?task)
                                   (exists (?x ?y)
                                     (and (member ?x (role-value
                                                         ?task 'has-jobs))
                                           (member ?y (role-value
                                                         ?task
                                                         'has-resources)))))))
   (has-goal-expression :type binary-kappa-expression
                        ;;;The goal is to generate a schedule
                        :default-value (kappa (?task ?schedule-model)
                                             (default-schedule-solution
                                              ?schedule-model ?task))))
  ;;there has to be at least one job and one resource to generate a schedule.
  :constraint (and (> (length (role-value ?task 'has-jobs)) 0)
                  (> (length (role-value ?task 'has-resources)) 0)))

;;;;;;;;;;;;;
```

```

(def-class JOB () ?j
  "A job is an entity that can be assigned to resources and time ranges and has a list of
  activities."
  ((has-activities :type list
    :documentation "Each job can have list of activities
    in order to accomplish the job.")
   (requires-resource :type resource :min-cardinality 1
    :documentation "It says that each job requires resources
    on which it can be assigned.")
   (requires-resource-type :type resource-type :min-cardinality 1)
   (has-time-range :type job-time-range :max-cardinality 1
    :documentation "It represents the time range of a job
    within which job must finish.")
   (has-due-date :type calendar-date :max-cardinality 1
    :documentation "It represents the calendar date of
    each job by which it has to dispatch.")
   (has-duration :type duration :max-cardinality 1)
   (has-load :type integer :default-value 1
    :documentation "It represents the number of resources that are
    needed by each job."))
  :iff-def (exists ?task (and (scheduling-task ?task)
    (member ?j (role-value ?task has-jobs)))))

(def-relation ASSIGNED-TO-RESOURCE (?j ?r ?sc)
  :iff-def (and (job ?j)
    (resource ?r)
    (schedule-model ?sc)
    (element-of (?j ?r ?a ?jtr) ?sc))
  :constraint (or (member ?r (setofall ?r2 (requires-resource ?j ?r2)))
    (empty-set (setofall ?r (requires-resource ?j ?r))))))

(def-relation ASSIGNED-TO-RESOURCE-TYPE (?j ?rtype ?sc)
  :iff-def (and (job ?j)
    (resource-type ?rtype)
    (schedule-model ?sc)
    (element-of (?j ?r ?a ?jtr) ?sc))
  :constraint (or (and (member ?rtype
    (setofall ?rtype2 (requires-resource-type
      ?j ?rtype2)))
    (holds ?rtype ?r))
    (empty-set ?rtype (requires-resource-type ?j ?rtype))))

(def-relation ASSIGNED-TO-JOB-TIME-RANGE (?j ?jtr ?sc)
  :iff-def (and (job ?j)
    (job-time-range ?jtr)
    (schedule-model ?sc)
    (element-of (?j ?r ?a ?jtr) ?sc))
  :constraint (or (member ?jtr (the ?jtr2 (has-time-range ?j ?jtr2)))
    (empty-set (the ?jtr (has-time-range ?j ?jtr)))))

(def-relation ASSIGNED-TO-ACTIVITY (?j ?a ?sc)
  :iff-def (and (job ?j)
    (activity ?a)
    (schedule-model ?sc)
    (element-of (?j ?r ?a ?jtr) ?sc))
  :constraint (or (member ?a (setofall ?a2 (has-activities ?j ?a2)))
    (empty-set (setofall ?a (has-activities ?j ?a)))))

(def-relation ASSIGNED-JOB (?x ?sc)
  "The job is said to be an assigned job if it is assigned to the resource and has a time
  range."
  :iff-def (and (exists ?r (and (resource ?r)
    (assigned-to-resource ?x ?r ?sc)))
    (exists ?a (and (activity ?a)
    (assigned-to-activity ?x ?a ?sc)))
    (exists ?jtr (and (job-time-range ?jtr)
    (assigned-to-job-time-range ?x ?jtr ?sc)))))

(def-relation UNASSIGNED-JOB (?x ?sc)
  "It is true if the job is not assigned-job."
  :iff-def (not (assigned-job ?x ?sc)))

(def-function RESOURCE-ASSIGNED-TO-A-JOB (?x ?sc) -> ?r
  "This function gives the resource assigned to the job in a schedule."
  :constraint (and (schedule-model ?sc)
    (resource ?r)
    (job ?x))
  :body (the ?r (assigned-to-resource ?x ?r ?sc)))

```

```

(def-function RESOURCE-TYPE-ASSIGNED-TO-A-JOB (?x ?sc) -> ?rtype
"This function gives the resource-type assigned to the job in a schedule."
:constraint (and (resource-type ?rtype)
                 (schedule-model ?sc)
                 (job ?x))
:body (the ?rtype (assigned-to-resource-type ?x ?rtype ?sc)))

(def-function TIME-RANGE-ASSIGNED-TO-A-JOB (?x ?sc) -> ?jtr
"This function gives a time-range assigned to the job in a schedule."
:constraint (and (schedule-model ?sc)
                 (job-time-range ?jtr)
                 (job ?x))
:body (the ?jtr (assigned-to-job-time-range ?x ?jtr ?sc)))

(def-function ACTIVITY-ASSIGNED-TO-A-JOB (?x ?sc) -> ?a
:constraint (and (schedule-model ?sc)
                 (activity ?a)
                 (job ?x))
:body (the ?a (assigned-to-activity ?x ?a ?sc)))

;;;;

(def-class JOB-TIME-RANGE () ?jtr
"It represents the time range of each job in terms of its earliest and latest start and end time."
((has-earliest-start-time :type time-point :min-cardinality 1)
 (has-latest-start-time :type time-point :min-cardinality 1)
 (has-earliest-end-time :type time-point :min-cardinality 1)
 (has-latest-end-time :type time-point :min-cardinality 1))
:constraint (or (precedes (the ?est (has-earliest-start-time ?jtr ?est))
                          (the ?eet (has-earliest-end-time ?jtr ?eet)))
                (precedes (the ?lst (has-latest-start-time ?jtr ?lst))
                          (the ?let (has-latest-end-time ?jtr ?let)))))

(def-relation JOB-START-TIME-EARLIER-THAN (?est1 ?est2)
"This relation states that if the earliest start time of job-time-range-1 is earlier than that of the other."
:constraint (and (time-point ?est1)
                 (time-point ?est2))
:iff-def (exists ?job (and (job ?job has-time-range ?jtr1)
                          (has-earliest-start-time ?jtr1 ?est1)
                          (exists ?job2 (and (job ?job2 has-time-range ?jtr2)
                                           (has-earliest-start-time ?jtr2 ?est2)
                                           (precedes ?est1 ?est2))))))

(def-relation JOB-TIME-RANGES-OVERLAP (?jtr-1 ?jtr-2)
"This overlapping relation is exclusively defined for the job time ranges."
:constraint (and (job-time-range ?jtr-1)
                 (job-time-range ?jtr-2))
:iff-def (and (and (precedes (the ?est-1 (has-earliest-start-time ?jtr-1 ?est-1))
                              (the ?est-2 (has-earliest-start-time ?jtr-2 ?est-2)))
                 (precedes (the ?lst-1 (has-latest-start-time ?jtr-1 ?lst-1))
                              (the ?lst-2 (has-latest-start-time ?jtr-2 ?lst-2))))
            (and (follows (the ?eet-1 (has-earliest-end-time ?jtr-1 ?eet-1))
                          (the ?est-2 (has-earliest-start-time ?jtr-2 ?est-2)))
                 (follows (the ?eet-1 (has-earliest-end-time ?jtr-1 ?eet-1))
                          (the ?lst-2 (has-latest-start-time ?jtr-2 ?lst-2))))
            (follows (the ?let-1 (has-latest-end-time ?jtr-1 ?let-1))
                      (the ?est-2 (has-earliest-start-time ?jtr-2 ?est-2)))
            (follows (the ?let-1 (has-latest-end-time ?jtr-1 ?let-1))
                      (the ?lst-2 (has-latest-start-time ?jtr-2 ?lst-2))))
            (and (precedes (the ?eet-1 (has-earliest-end-time ?jtr-1 ?eet-1))
                          (the ?eet-2 (has-earliest-end-time ?jtr-2 ?eet-2)))
                 (precedes (the ?let-1 (has-latest-end-time ?jtr-1 ?let-1))
                          (the ?let-2 (has-latest-end-time ?jtr-2 ?let-2))))))

(def-function START-TIME-OF-A-JOB (?j ?jtr) -> ?est
"This function retrieves the earliest start time of each job."
:constraint (and (job ?j has-time-range ?jtr)
                 (time-point ?est))
:body (the ?est (has-earliest-start-time ?jtr ?est)))

(def-function LATEST-START-TIME-OF-A-JOB (?j ?jtr) -> ?let
"This function retrieves the latest start time of each job."
:constraint (and (job ?j has-time-range ?jtr)
                 (time-point ?let))
:body (the ?let (has-latest-end-time ?jtr ?let)))

```

```

(def-function EARLIEST-END-TIME-OF-A-JOB (?j ?jtr) -> ?eet
  "This function retrieves an earliest end time of each job."
  :constraint (and (job ?j has-time-range ?jtr)
                   (time-point ?eet))
  :body (the ?eet (has-earliest-end-time ?jtr ?eet)))

(def-function LATEST-END-TIME-OF-A-JOB (?j ?jtr) -> ?let
  "This function retrieves the latest end time of each job."
  :constraint (and (job ?j has-time-range ?jtr)
                   (time-point ?let))
  :body (the ?let (has-latest-end-time ?jtr ?let)))

(def-function JOB-TIME-RANGE-DURATION (?j ?jtr) -> ?time-point
  "This function calculates the duration of a job."
  :constraint (and (job ?j)
                   (has-time-range ?j ?jtr)
                   (job-time-range ?jtr))
  :body (- (the-slot-value ?jtr has-latest-end-time)
           (the-slot-value ?jtr has-earliest-start-time)))

(def-class JOB-TYPE (job) ?jt
  ((has-activity-type :type activity-type
                     :documentation "It specialises an activity
                                     in its more specific types."))
  :iff-def (subclass-of ?jt job))

(def-function DUE-DATE-OF-A-JOB (?j) -> ?due-date
  "This function returns a due-date of a job."
  :constraint (and (calendar-date ?due-date)
                   (job ?j))
  :body (the ?due-date (has-due-date ?j ?due-date)))

(def-relation JOB-PRECEDES (?j1 ?j2)
  "This relation expresses the temporal constraint among any two jobs says that, if the
  latest-end-time of j1 is less than or equal to the earliest-start-time of j2 then j1
  precedes j2."
  :constraint (and (job ?j1)
                   (job ?j2))
  :iff-def (and (has-time-range ?j1 ?jtr-1)
                (has-time-range ?j2 ?jtr-2)
                (not (= (?j1 ?j2)))
                (<= (the-slot-value ?jtr-1 has-latest-end-time)
                     (the-slot-value ?jtr-2 has-earliest-start-time))))

(def-relation CRITICAL-JOB (?j1 ?j2)
  "The job-1 is a critical job as that of job-2, if the due-date of job-1 is earlier than
  due-date job-2."
  :iff-def (and (job ?j1 has-due-date ?dd1)
                (job ?j2 has-due-date ?dd2)
                (not (= (?j1 ?j2)))
                (due-date-earlier-than-other ?dd1 ?dd2)))

(def-relation HIGHER-PRIORITY-JOB (?j1 ?j2)
  "This relation states that if a job-time-range-duration of one job is less than that of a
  other job then the job is a higher priority job."
  :constraint (and (job ?j1 has-time-range ?jtr1)
                   (job ?j2 has-time-range ?jtr2))
  :iff-def (and (= (job-time-range-duration ?j1 ?jtr1) ?jd1)
                (= (job-time-range-duration ?j2 ?jtr2) ?jd2)
                (< ?jd1 ?jd2)))

(def-relation HIGHER-PRIORITY-JOB-BASED-ON-ACTIVITIES (?j1 ?j2)
  "This relation states that a job that has a least number of activities is a high priority
  job."
  :constraint (and (job ?j1 has-activities ?a1)
                   (job ?j2 has-activities ?a2))
  :iff-def (and (= (number-of-activities-within-job ?j1) ?list1)
                (= (number-of-activities-within-job ?j2) ?list2)
                (= (length ?list1) ?l1)
                (= (length ?list2) ?l2)
                (< ?l1 ?l2)))

(def-relation EARLIER-START-TIME-OF-A-JOB (?j1 ?j2)
  :constraint (and (job ?j1 has-time-range ?jtr1)
                   (job ?j2 has-time-range ?jtr2))
  :iff-def (and (has-earliest-start-time ?jtr1 ?est1)
                (has-earliest-start-time ?jtr2 ?est2)
                (<> ?j1 ?j2)
                (job-start-time-earlier-than ?est1 ?est2)))

```

```

(def-relation ACTIVITY-PRECEDES (?a1 ?a2)
:constraint (and (activity ?a1)
                 (activity ?a2))
:iff-def (and (has-time-range ?a1 ?jtr-a1)
              (has-time-range ?a2 ?jtr-a2)
              (has-duration ?a1 ?d1)
              (<= (time-point-sum (the-slot-value
                                   ?jtr-a1 has-earliest-start-time)
                       (magnitude-of-duration ?d1))
                  (the-slot-value ?jtr-a2 has-earliest-start-time)))
:axiom-def (defines-partial-order activity-precedes))

;;;;;;;;;;
;Temporal relations among jobs;
;;;;;;;;;;

(def-relation FINISHES-BEFORE (?j1 ?j2)
"This relation says that if earliest end time of of job-1 precedes the earliest start time
of job-2, then job-1 finishes-before job-2."
:constraint (and (job ?j1 has-time-range ?jtr1)
                 (job ?j2 has-time-range ?jtr2))
:iff-def (precedes (the-slot-value ?jtr1 has-earliest-end-time)
              (the-slot-value ?jtr2 has-latest-start-time)))

(def-relation JOB1-SCHEDULED-BEFORE-JOB2 (?j1 ?j2)
:constraint (and (job ?j1 has-time-range ?jtr1)
                 (job ?j2 has-time-range ?jtr2)
                 (= ?jd1 (job-time-range-duration ?j1 ?jtr1))
                 (= ?jd2 (job-time-range-duration ?j2 ?jtr2)))
:iff-def (and (< (+ ?jd1 ?jd2)
                 (- (latest-end-time ?jtr1 ?let1)
                    (earliest-start-time ?jtr2 ?est1)))
            (<= (+ ?jd1 ?jd2)
                 (- (latest-end-time ?jtr2 ?let2)
                    (earliest-start-time ?jtr1 ?est1))))))

(def-relation JOB2-SCHEDULED-BEFORE-JOB1 (?j2 ?j1)
:constraint (and (job ?j1 has-time-range ?jtr1)
                 (job ?j2 has-time-range ?jtr2)
                 (= ?jd1 (job-time-range-duration ?j1 ?jtr1))
                 (= ?jd2 (job-time-range-duration ?j2 ?jtr2)))
:iff-def (and (< (+ ?jd1 ?jd2)
                 (- (latest-end-time ?jtr2 ?let2)
                    (earliest-start-time ?jtr1 ?est1)))
            (<= (+ ?jd1 ?jd2)
                 (- (latest-end-time ?jtr1 ?let1)
                    (earliest-start-time ?jtr2 ?est2))))))

(def-relation NO-FEASIBLE-ORDERING-POSSIBLE (?j1 ?j2)
:constraint (and (job ?j1 has-time-range ?jtr1)
                 (job ?j2 has-time-range ?jtr2)
                 (= ?jd1 (job-time-range-duration ?j1 ?jtr1))
                 (= ?jd2 (job-time-range-duration ?j2 ?jtr2)))
:iff-def (and (> (+ ?jd1 ?jd2)
                 (- (latest-end-time ?jtr1 ?let1)
                    (earliest-start-time ?jtr2 ?est2)))
            (> (+ ?jd1 ?jd2)
                 (- (latest-end-time ?jtr2 ?let2)
                    (earliest-start-time ?jtr1 ?est1))))))

(def-relation ANY-ORDERING-IS-ALLOWED (?j1 ?j2)
:constraint (and (job ?j1 has-time-range ?jtr1)
                 (job ?j2 has-time-range ?jtr2)
                 (= ?jd1 (job-time-range-duration ?j1 ?jtr1))
                 (= ?jd2 (job-time-range-duration ?j2 ?jtr2)))
:iff-def (and (<= (+ ?jd1 ?jd2)
                  (- (latest-end-time ?jtr1 ?j1)
                     (earliest-start-time ?jtr2 ?j2)))
            (<= (+ ?jd1 ?jd2)
                  (- (latest-end-time ?jtr2 ?j2)
                     (earliest-start-time ?jtr1 ?j1))))))

;;;;;;;;;;

(def-function NUMBER-OF-ACTIVITIES-WITHIN-JOB (?j) -> ?list
"This function retrieves the list of activities within each job."
:constraint (and (job ?j)
                 (list ?list))
:body (the ?list (has-activities ?job ?list)))

```

```

(def-relation JOB-FINISHES-IN-TIME (?j ?sc)
:constraint (and (job ?j has-time-range ?jtr)
                 (has-due-date ?j ?dd)
                 (schedule-model ?sc)
                 (element-of (?j ?r ?a ?jtr) ?sc))
:iff-def (or (< (the-slot-value ?jtr has-latest-end-time)
               (the-slot-value ?dd day-of))
           (< (the-slot-value ?jtr has-latest-end-time)
               (the-slot-value ?dd month-of))
           (< (the-slot-value ?jtr has-latest-end-time)
               (the-slot-value ?dd year-of))))

(def-relation JOB-NOT-FINISHES-IN-TIME (?job ?sc)
:constraint (and (job ?job)
                 (schedule-model ?sc))
:iff-def (not (job-finishes-in-time ?job ?sc)))

(def-function LATENESS-OF-A-JOB-BY-DAY (?j) -> ?tp
:constraint (and (job ?j)
                 (has-time-range ?j ?jtr)
                 (has-due-date ?j ?ddj)
                 (time-point ?tp))
:body (the ?tp (- (the-slot-value ?jtr has-latest-end-time)
                  (the-slot-value ?ddj day-of))))

(def-function LATENESS-OF-A-JOB-BY-MONTH (?j) -> ?tp
:constraint (and (job ?j)
                 (has-time-range ?j ?jtr)
                 (has-due-date ?j ?ddj)
                 (time-point ?tp))
:body (the ?tp (- (the-slot-value ?jtr has-latest-end-time)
                  (the-slot-value ?ddj month-of))))

(def-function LATENESS-OF-A-JOB-BY-YEAR (?j) -> ?tp
:constraint (and (job ?j)
                 (has-time-range ?j ?jtr)
                 (has-due-date ?j ?ddj)
                 (time-point ?tp))
:body (the ?tp (- (the-slot-value ?jtr has-latest-end-time)
                  (the-slot-value ?ddj year-of))))

(def-function JOB-TARDINESS (?j) -> ?tp
:constraint (and (job ?j)
                 (time-point ?tp))
:body (the ?tp (lateness-of-a-job-by-day ?j)))

(def-function JOB-TARDINESS-FOR-A-MONTH (?j) -> ?tp
:constraint (and (job ?j)
                 (time-point ?tp))
:body (the ?tp (lateness-of-a-job-by-month ?j)))

(def-function JOB-TARDINESS-FOR-A-YEAR (?j) -> ?tp
:constraint (and (job ?j)
                 (time-point ?tp))
:body (the ?tp (lateness-of-a-job-by-year ?j)))

;;;;;;;;;;;;;;

(def-class ACTIVITY () ?a
  "It represents the list of activities within a job."
  ((has-duration :type duration :max-cardinality 1
                 :documentation "This represents a duration of
                                an individual activity.")
   (requires-resource :type resource :cardinality 1)
   (requires-resource-type :type resource-type :cardinality 1)
   (has-job-belonging :type job :cardinality 1
                      :documentation "This represents a job to which an
                                      activity belongs.")
   (has-time-range :type job-time-range :max-cardinality 1
                   :documentation "It represents the time range
                                   of each activity.")
   (has-load :type integer :default-value 1))
:iff-def (exists ?j (and (job ?j)
                         (member ?a (has-activities ?j ?list))))
:constraint (exists ?task (and (scheduling-task ?task)
                              (member ?a (role-value ?task has-activities))))

```

```

(def-class RESOURCE () ?r
  "The resource is an entity to which the jobs can be assigned for their execution."
  ((handles-job-type :type job-type :cardinality 1
    :documentation "It represents the type of jobs
      each resource is capable of handling.")
   (handles-job :type job :cardinality 1
    :documentation "It represents the kind of jobs each
      resource is capable of handling.")
   (handles-activity :type activity :cardinality 1
    :documentation "It represents the kind of activities
      each resource is capable of handling.")
   (has-availability :type time-range :min-cardinality 1
    :documentation "It represents the availability period
      of each resource within which resource
      can execute the jobs.")
   (has-capacity :type number :default-value 1
    :documentation "It represents the number of jobs each
      resource can handle in parallel.)))

:iff-def (exists ?task (and (scheduling-task ?task)
  (member ?r (role-value ?task has-resources))))
:constraint (or (exists ?jtype (and (job-type ?jtype)
  (handles-job-type ?r ?jtype)))
  (exists ?j (and (job ?j)
    (handles-job ?r ?j)))
  (exists ?a (and (activity ?a)
    (handles-activity ?r ?a)))))

(def-class UNARY-RESOURCE (resource) ?ur
  :constraint (exists ?r (and (resource ?r)
    (= (MAXIMUM-CAPACITY-OF-RESOURCE ?r) 1))))

(def-axiom RESOURCE-CAPACITY
  "This axiom says that if there is a resource ri which has capacity ni, then schedule cannot
  have more than ni jobs whose time ranges are overlapping with each other."
  (forall (?ri ?sc)
    (=> (unary-resource ?ri has-capacity ?ni)
      (not (exists ?j (and (element-of (?j ?ri ?a ?jtr) ?sc)
        (= ?all (setofall ?j2
          (and (element-of (?j2 ?ri ?a2 ?jtr2) ?sc)
            (job-time-ranges-overlap
              (?jtr ?jtr2))
            (not (= (?j ?j2))))))
        (> (length (cons ?j ?all2)) ?ni)))))))

(def-class RESOURCE-TYPE () ?rt
  :iff-def (subclass-of ?rt resource))

(def-relation JOB-AND-RESOURCE-TIME-RANGE (?j ?r)
  "This relation states that a time range of a job should be during or equal to the
  availability period of a resource."
  :constraint (and (job ?j requires-resource ?r)
    (has-time-range ?j ?jtr)
    (resource ?r has-availability ?tr))
  :iff-def (job-time-range-during-or-equal ?jtr ?tr))

(def-function RESOURCE-HANDLES-JOB (?r ?sc) -> ?j
  "This function retrieves the jobs that a resource can handle in a schedule."
  :constraint (and (resource ?r)
    (schedule-model ?sc)
    (job ?j))
  :body (the ?j (handles-job ?r ?j ?sc)))

(def-function RESOURCE-TIME-AVAILABILITY (?r) -> ?tr
  "This function retrieves the time-availability of a resource."
  :constraint (time-range ?tr)
  :body (the ?tr (resource ?r has-availability ?tr)))

(def-function MAXIMUM-CAPACITY-OF-RESOURCE (?r) -> ?number
  "This function gives the maximum capacity of a resource."
  :constraint (and (resource ?r)
    (number ?number))
  :body (the ?number (has-capacity ?r ?number)))
  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(def-class CONSTRAINT () ?c
  "This definition of constraint is common to the hard constraint as well."
  ((applicability-condition :default-value (kappa (?sc) (true)) :type unary-relation)
   (has-expression :cardinality 1 :type unary-relation
                    :documentation "This argument is a schedule"))))

(def-class HARD-CONSTRAINT (constraint))

(def-class REQUIREMENT ()
  "A requirement express the properties which has to be satisfied by a solution schedule."
  ((applicability-condition :default-value (kappa (?sc) (true))
                           :type unary-relation)
   (has-expression :cardinality 1 :type unary-relation
                    :documentation "The argument must be a schedule"))))

(def-relation REQUIREMENT-APPLICABLE (?r ?sc)
  :constraint (requirement ?r)
  :iff-def (holds (the ?x (applicability-condition ?r ?x)) ?sc))

(def-relation CONSTRAINT-APPLICABLE (?c ?sc)
  :constraint (constraint ?c)
  :iff-def (holds (the ?x (applicability-condition ?c ?x)) ?sc))

(def-relation HARD-CONSTRAINT-APPLICABLE (?hc ?sc)
  :constraint (hard-constraint ?hc)
  :iff-def (holds (the ?x (applicability-condition ?hc ?x)) ?sc))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;NOTE: Classes Time-Range, Duration are defined in the Simple-Time ontology.;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(def-class SCHEDULE-MODEL (set) ?schedule-task
  "The schedule is defined in terms of a quadruple of the form (job resource activity job-
  time-range) which is modelled by the class job-assignment. The schedule is true for any
  element of class job-assignment and false for any other quadruple."
  :iff-def (and (= ?quadruples (setofall ?quadruple
                                           (element-of ?quadruple ?schedule-task)))
               (every ?quadruples job-assignment)))

(def-class JOB-ASSIGNMENT () ?quadruple
  "The job assignment models a quadruple of the form (job resource activity job-time-range)."
  :iff-def (and (== ?quadruple (?j ?r ?a ?jtr))
                (job ?j)
                (member ?a (has-activities ?j ?list))
                (resource ?r)
                (job-time-range ?jtr)))

(def-class SCHEDULE-TYPE () ?c
  :iff-def (subclass-of ?c schedule-model))

(def-relation DEFAULT-SCHEDULE-SOLUTION (?sc ?task)
  :constraint (and (schedule-model ?sc)
                  (scheduling-task ?task))
  :iff-def (and (schedule-is-correct ?sc)
                (schedule-minimally-complete ?sc
                                              (role-value ?task has-jobs))
                (maximally-admissible-schedule ?sc
                                                  (role-value
                                                    ?task has-hard-constraints)
                                                  (role-value ?task has-requirements))))
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(def-relation SCHEDULE-IS-CORRECT (?sc)
  "It says that if no pair <j . a> appears in more than one quadruple in a schedule."
  :iff-def (and (schedule-model ?sc)
                (= (setofall (?j . ?a)
                             (element-of (?j ?r ?a ?jtr) ?sc)) ?quadruple1)
                (not (exists ?quadruple2
                              (and (element-of ?quadruple2 ?sc)
                                   (member (?j . ?a) ?quadruple2))))))

(def-relation SCHEDULE-MINIMALLY-COMPLETE (?sc ?jobs)
  "The schedule is complete when each job is assigned to a resource and has a time range."
  :iff-def (not (exists ?x
                        (and (member ?x ?jobs)
                             (unassigned-job ?x ?sc)))))

```

```

(def-relation SCHEDULE-MINIMALLY-INCOMPLETE (?sc ?jobs)
  :iff-def (exists ?x (and (member ?x ?jobs)
                           (unassigned-job ?x ?sc))))

(def-relation SCHEDULE-IS-FEASIBLE (?sc ?requirements)
  "The schedule is feasible if it satisfies all the requirements by the completion of a
  schedule."
  :constraint (and (list ?requirements)
                   (every ?requirements requirement)
                   (schedule-model ?sc))
  :iff-def (not (exists ?x (and (member ?x ?requirements)
                                (schedule-violates-requirement ?sc ?x)))))

(def-relation SCHEDULE-VIOLATES-REQUIREMENT (?sc ?requirements)
  :constraint (and (list ?requirements)
                   (every ?requirements requirement)
                   (schedule-model ?sc))
  :iff-def (and (requirement-applicable ?requirements ?sc)
                (not (holds (the ?x (has-expression ?requirements ?x)) ?sc))))

(def-relation SCHEDULE-SATISFIES-REQUIREMENT (?sc ?requirements)
  :constraint (and (list ?requirements)
                   (every ?requirements requirement)
                   (schedule-model ?sc))
  :iff-def (and (requirement-applicable ?requirements ?sc)
                (holds (the ?x (has-expression ?requirements ?x)) ?sc)))

(def-relation MINIMALLY-ADMISSIBLE-SCHEDULE (?sc ?hard-constraints)
  "The schedule is minimally admissible if none of the hard constraints are violated."
  :constraint (and (list ?hard-constraints)
                   (every ?hard-constraints hard-constraint)
                   (schedule-model ?sc))
  :iff-def (not (exists ?x
                      (and (member ?x ?hard-constraints)
                           (schedule-violates-constraint ?sc ?x)))))

(def-relation MAXIMALLY-ADMISSIBLE-SCHEDULE (?sc ?hard-constraints)
  "The schedule is maximally admissible if it satisfies all the hard as well as soft
  constraints by the completion of a schedule."
  :constraint (and (list ?hard-constraints)
                   (every ?hard-constraints hard-constraint)
                   (schedule-model ?sc))
  :iff-def (not (exists ?x
                      (and (member ?x ?hard-constraints)
                           (schedule-violates-constraint ?sc ?x)))))

(def-relation SCHEDULE-VIOLATES-CONSTRAINT (?sc ?constraints)
  :constraint (and (list ?constraints)
                   (every ?constraints constraint)
                   (schedule-model ?sc))
  :iff-def (and (constraint-applicable ?constraints ?sc)
                (not (holds (the ?x (has-expression ?constraints ?x)) ?sc))))

(def-relation SCHEDULE-SATISFIES-CONSTRAINT (?sc ?constraints)
  :constraint (and (list ?constraints)
                   (every ?constraints constraint)
                   (schedule-model ?sc))
  :iff-def (and (constraint-applicable ?constraints ?sc)
                (holds (the ?x (has-expression ?constraints ?x)) ?sc)))

(def-relation SCHEDULE-SATISFIES-HARD-CONSTRAINT (?sc ?hard-constraints)
  :constraint (and (list ?hard-constraints)
                   (every ?hard-constraints hard-constraint)
                   (schedule-model ?sc))
  :iff-def (and (hard-constraint-applicable ?hard-constraints ?sc)
                (holds (the ?x (has-expression ?hard-constraints ?x)) ?sc)))

(def-relation ADMISSIBLE-SCHEDULE (?js ?as)
  "This relation says that the time range of each activity within a job has to be DURING the
  time range of a job."
  :constraint (and (list ?js)
                   (every ?js job)
                   (list ?as)
                   (every ?as activity)
                   (has-time-range ?js ?jtr)
                   (has-time-range ?as ?jtra))
  :iff-def (job-activity-time-range-during ?jtra ?jtr))

```

```

(def-relation OPTIMAL-SCHEDULE-SOLUTION (?sc1 ?task)
"The schedule-solution Ssol is an optimal if there is no other schedule solution which has
a lower cost than Ssol."
:constraint (scheduling-task ?task)
:iff-def (and (default-schedule-solution ?sc1 ?task)
(not (exists ?sc2
(and (default-schedule-solution ?sc2 ?task)
(has-cost-order-relation ?task ?rel)
(cheaper-schedule ?rel ?sc1 ?sc2))))))

(def-relation CHEAPER-SCHEDULE (?rel ?sc1 ?sc2)
:constraint (and (order-relation ?rel)
(schedule-model ?sc1)
(schedule-model ?sc2))
:iff-def (holds ?rel1 ?sc1 ?sc2))

(def-relation SCHEDULE-EXTENDS (?sc1 ?sc2)
:iff-def (and (forall ?j
(=> (assigned-job ?j ?sc2)
(assigned-job ?j ?sc1)))
(exists ?j2 (and (assigned-job ?j2 ?sc1)
(unassigned-job ?j2 ?sc2)))))

;;;;;;;;;;;;;

(def-class PREFERENCE () ?p
"A preference gives the order over two schedules."
((has-expression :cardinality 1 :type prefer-expression)))

(def-class PREFER-EXPRESSION (proof-expression) ?exp
((proves-relation :value prefer)
:constraint (and (== ?exp (?tail if . ?rest))
(== ?tail (prefer ?schedule-task1 ?schedule-task2)))))

(def-relation PREFER (?schedule-task1 ?schedule-task2)
"This relation expresses the preferences between two schedules."
:constraint (and (schedule-model ?schedule-task1)
(schedule-model ?schedule-task2))
:axiom-def (defines-partial-order prefer))

(def-axiom COST-SUBSUMES-PREFERENCES
"This axiom tells that the cost function subsumes each preference."
(forall (?schedule-task1 ?schedule-task2)
(=>
(and (scheduling-task ?task has-preferences ?prs
has-cost-function ?cf)
(has-cost-order-relation ?task ?rel)
(member ?pr ?prs)
(has-expression ?pr ?exp)
(proves ?exp `(prefer ?schedule-task1 ?schedule-task2)))
(cheaper-schedule ?rel ?schedule-task1 ?schedule-task2))))

(def-axiom COST-PREFERENCE-CONSISTENCY
"This axiom states that the cost function should not contradict any partial order expressed
by preferences."
(forall (?schedule-task1 ?schedule-task2)
(=> (and (scheduling-task ?task has-preferences ?prs
has-cost-function ?cf)
(has-cost-order-relation ?task ?rel)
(cheaper-schedule ?rel ?schedule-task1 ?schedule-task2))
(not (exists ?pr
(member ?pr ?prs)
(has-expression ?pr ?exp)
(proves ?exp `(prefer
?schedule-task2 ?schedule-task1)))))))

(def-class COST () ?x
"The cost is represented as a Real-Number or Vector."
:sufficient (or (real-number ?x)
(vector ?x)))

(def-class COST-FUNCTION (unary-function) ?cf
"This function takes a schedule as an input and returns its cost."
:iff-def (and (domain ?cf schedule-model)
(range ?cf cost)))

(def-relation HAS-COST-ORDER-RELATION (?scheduling-task ?rel)
:iff-def (= ?rel (third (has-cost-algebra ?scheduling-task ?alg))))

(def-relation HAS-COST-DIFFERENCE-FUNCTION (?scheduling-task ?rel)
:iff-def (= ?rel (second (has-cost-algebra ?scheduling-task ?alg))))

```

```

(def-relation HAS-COST-SUM-FUNCTION (?scheduling-task ?rel)
  :iff-def (= ?rel (first (has-cost-algebra ?scheduling-task ?alg))))

(def-function ADD-VECTOR-COSTS (?c1 &rest ?rest-costs)
  :constraint (and (= (length ?c1) ?n)
    (every ?rest-costs (kappa (?c)
      (= (length ?c) ?n))))
  :body (if (null ?c1)
    nil
    (cons (apply + (map first (cons (?c1 ?rest-costs)))
      (apply add-vector-costs
        (map rest (cons ?c1 ?rest-costs)))))
      )))

(def-function SUBTRACT-VECTOR-COSTS (?c1 &rest ?rest-costs)
  :constraint (and (= (length ?c1) ?n)
    (every ?rest-costs (kappa (?c)
      (= (length ?c) ?n))))
  :body (if (null ?c1)
    nil
    (cons (apply - (map first (cons (?c1 ?rest-costs)))
      (apply add-vector-costs
        (map rest (cons ?c1 ?rest-costs)))))
      )))

(def-relation CHEAPER-VECTOR-COST (?c1 ?c2)
  :iff-def (and (not (null ?c1))
    (not (null ?c2))
    (or (< (first ?c1)
      (first ?c2))
      (cheaper-vector-cost (rest ?c2) (rest ?c2)))))

;;;;;;;;;;;;;
;Rather loosely constrained job precedence relations;
;;;;;;;;;;;;;

(def-relation STARTS-AFTER (?j1 ?j2)
  "This relation is opposite of finishes-before, it implies, if earliest start time of job-2
  follows the latest end time of job-1, then job-2 starts-after job-1."
  :constraint (and (job ?j1 has-time-range ?jtr1)
    (job ?j2 has-time-range ?jtr2))
  :iff-def (follows (the-slot-value ?jtr2 has-earliest-start-time)
    (the-slot-value ?jtr1 has-latest-end-time)))

(def-relation EQUALS (?j1 ?j2)
  "This relation says that both jobs job-1 and job-2 are equal to each other, if they start
  simultaneously and finish simultaneously."
  :constraint (and (job ?j1 has-time-range ?jtr1)
    (job ?j2 has-time-range ?jtr2))
  :iff-def (and (time-points-equal (the-slot-value ?jtr1 has-earliest-start-time)
    (the-slot-value ?jtr2 has-earliest-start-time))
    (time-points-equal (the-slot-value ?jtr1 has-latest-end-time)
      (the-slot-value ?jtr2 has-latest-end-time))))

(def-relation JOB-MEETS (?j1 ?j2)
  :constraint (and (job ?j1 has-time-range ?jtr1)
    (job ?j2 has-time-range ?jtr2))
  :iff-def (time-points-equal (the-slot-value ?jtr1 has-latest-end-time)
    (the-slot-value ?jtr2 has-earliest-start-time)))

(def-relation JOBS-OVERLAP (?j1 ?j2)
  :constraint (and (job ?j1 has-time-range ?jtr1)
    (job ?j2 has-time-range ?jtr2))
  :iff-def (and (precedes (the-slot-value ?jtr1 has-latest-start-time)
    (the-slot-value ?jtr2 has-earliest-start-time))
    (follows (the-slot-value ?jtr1 has-earliest-end-time)
      (the-slot-value ?jtr2 has-earliest-start-time))
    (precedes (the-slot-value ?jtr1 has-earliest-end-time)
      (the-slot-value ?jtr2 has-earliest-end-time))))

(def-relation JOB-IS-DURING (?j1 ?j2)
  :constraint (and (job ?j1 has-time-range ?jtr1)
    (job ?j2 has-time-range ?jtr2))
  :iff-def (and (follows (the-slot-value ?j1 has-earliest-start-time)
    (the-slot-value ?j2 has-latest-start-time))
    (precedes (the-slot-value ?j1 has-latest-end-time)
      (the-slot-value ?j2 has-earliest-end-time))))

(def-relation JOBS-START-SIMULTANEOUSLY (?j1 ?j2)
  :constraint (and (job ?j1 has-time-range ?jtr1)
    (job ?j2 has-time-range ?jtr2))
  :iff-def (time-points-equal (the-slot-value ?jtr1 has-earliest-start-time)
    (the-slot-value ?jtr2 has-earliest-start-time)))

```

```
(def-relation JOBS-FINISH-SIMULTANEOUSLY (?j1 ?j2)
:constraint (and (job ?j1 has-time-range ?jtr1)
                 (job ?j2 has-time-range ?jtr2))
:iff-def (time-points-equal (the-slot-value ?jtr1 has-latest-end-time)
                             (the-slot-value ?jtr2 has-latest-end-time)))

;;;;;;;;;
```

## Appendix 2

# A COMPLETE SPECIFICATION OF A GENERIC MODEL OF SCHEDULING PROBLEM-SOLVING

---

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Base: 10; Package: OCML; -*-

;;;THE OPEN UNIVERSITY

;;;Author: Dnyanesh Rajpathak

(in-package "OCML")

(in-ontology generic-schedule)

(def-class SCHEDULE-SPACE () ?x
  "The schedule space is composed of set of schedule states and it is associated with the
  scheduling task."
  ((associated-with-task :type scheduling-task :cardinality 1)
   (has-states :type set :cardinality 1 :default-value nil))
  :constraint (=> (member ?s (the ?set (has-states ?x ?set)))
                (schedule-state ?s)))

(def-class SCHEDULE-STATE () ?c
  "Each schedule state has a unique association with a schedule model."
  ((has-schedule-model :cardinality 1 :type schedule-model)))

(def-relation STATE-TRANSITION (?s1 ?schedule-op ?s2)
  "This relation is essential to achieve the state transition within a schedule space."
  :iff-def (and (schedule-state ?s1 has-schedule-model ?schedule-task1)
                (schedule-state ?s2 has-schedule-model ?schedule-task2)
                (schedule-operator ?schedule-op has-body ?fun)
                (= ?schedule-task2 (call ?fun ?schedule-task1))
                (not (= ?schedule-task1 ?schedule-task2))))

(def-function PREDECESSOR (?s)
  "This function retrieves the predecessor state of a current state."
  :constraint (schedule-state ?s)
  :body (the ?s1 (state-transition ?s1 ?op ?s)))

(def-function SUCCESSOR (?s)
  "This function retrieves the successor state of a current state."
  :constraint (schedule-state ?s)
  :body (the ?s1 (state-transition ?s ?op ?s1)))

(def-function COMPUTE-STATE-COST (?s ?task)
  :constraint (and (schedule-state ?s)
                  (scheduling-task ?task))
  :body (call (the ?f (has-cost-function ?task ?f))
              (the ?schedule-task (has-schedule-model ?s ?schedule-task))))

;;;;;;;;;;;;;

(def-class SCHEDULE-OPERATOR ()
  "A state transition in the problem-space specifies a link between
  two schedule states, that is to say between two schedules. State transitions
  are carried out by means of schedule operators."
  ((assumption :default-value (true)
               :type relation-expression
               :documentation
               "This slot can be used to specify a statement that is expected
               to hold for the application domain in which the operator is
               applied. The difference between assumptions and preconditions
               is that while the former are static and the latter are dynamic.
               The truth value of a precondition might change during the schedule
               generation. Assumptions may or are expected to remain (un-) satisfied
               during the scheduling process")
   (has-precondition :default-value (true)
                     :type relation-expression)
   (has-body :type schedule-operator-body)))
```

```

(def-class SCHEDULE-OPERATOR-BODY (unary-function) ?fun
  "A body of a schedule operator is a unary function that takes as input a schedule-model,
  says schedule-taski, and produces as an output a schedule-model schedule-taskj"
  :no-op (:constraint (and (domain ?fun schedule-model)
                           (range ?fun schedule-model)
                           (<=> (= (call ?fun ?schedule-taski) ?schedule-taskj)
                                (not (= ?schedule-taski ?schedule-taskj))))))

(def-class BASIC-OPERATOR (schedule-operator)
)

(def-class MULTIPLE-OPERATOR (schedule-operator)
)

(def-relation SCHEDULE-OPERATOR-ORDER (?x ?c)
  :constraint (and (schedule-operator ?x)
                  (schedule-operator ?c)
                  (not (= ?x ?c))))

(tell (defines-partial-order schedule-operator-order))

(def-function COMPUTE-OPERATOR-COST (?op ?task)
  :constraint (and (schedule-operator ?op)
                  (scheduling-task ?task))
  :body (if (and (has-cost-difference-function ?task ?fun)
                 (state-transition ?s1 ?op ?s2))
            (call ?fun ?s2 ?s1)))
;;;;;;;;;

(def-class SCHEDULE-EXTENSION-RESOURCE-OPERATOR (schedule-operator)
  "This operator can be used to assign a job to its resources."
  ((applicable-to-jobs :default-value '(setofall ?x (job ?x))
                       :type function-expression
                       :documentation "An expression which returns the set
                                     of jobs whose resources can be assigned
                                     by means of this operator")
   (has-precondition :default-value (kappa (?schedule-task) (true))
                    :type relation-expression
                    :documentation "This is an expression which can be used to check
                                     if an operator is applicable in the current
                                     state - i.e. schedule-model. This expression
                                     should not depend on a particular job")
   (has-body :type schedule-extension-resource-operator-body)))

(def-class SCHEDULE-EXTENSION-RESOURCE-OPERATOR-BODY (lambda-expression) ?x
  "A basic schedule extension operator body is a unary function which takes an
  unassigned job, say ?j, and returns a resource, ?r, which is assigned to ?j
  in the successor schedule state"
  :no-op (:constraint (and (nth-domain ?x 1 job)
                           (nth-domain ?x 2 ?sc)
                           (>= (= ?z (call ?x ?j))
                                (and (requires-resource ?j ?resource)
                                     (resource ?z))))))

(def-class SCHEDULE-EXTENSION-RESOURCE-TYPE-OPERATOR (schedule-operator)
  "This operator can be used to assign a job to its more specific resource types."
  ((applicable-to-jobs :default-value '(setofall ?x (job ?x))
                       :type function-expression
                       :documentation "An expression which returns the set
                                     of jobs whose resource-types can be
                                     assigned by means of this operator")
   (has-precondition :default-value (kappa (?schedule-task) (true))
                    :type relation-expression
                    :documentation "This is an expression which can be used to check
                                     if an operator is applicable in the current
                                     state - i.e. schedule. This expression
                                     should not depend on a particular job")
   (has-body :type schedule-extension-resource-type-operator-body)))

(def-class SCHEDULE-EXTENSION-RESOURCE-TYPE-OPERATOR-BODY (lambda-expression) ?x
  "A basic schedule extension operator body is a unary function which takes an
  unassigned job, say ?j and produces a result, ?z, which belongs to the
  resource-type. ?z is taken as the new resource-type of ?j in the successor
  schedule state"
  :no-op (:constraint (and (nth-domain ?x 1 job)
                           (nth-domain ?x 2 ?sc)
                           (>= (= ?z (call ?x ?j))
                                (and (requires-resource-type ?j ?resource-type)
                                     (resource-type ?z))))))

```

```

(def-class SCHEDULE-EXTENSION-TIME-RANGE-OPERATOR (schedule-operator)
  "This operator can be used to assign a job to its time range."
  ((applicable-to-jobs :default-value '(setofall ?x (job ?x))
    :type function-expression
    :documentation "An expression which returns the set
      of jobs whose resources can be assigned
      by means of this operator")
   (has-precondition :default-value (kappa (?schedule-task) (true))
    :type relation-expression
    :documentation "This is an expression which can be used to check
      if an operator is applicable in the current
      state - i.e. schedule. This expression
      should not depend on a particular job")
   (has-body :type schedule-extension-time-range-operator-body)))

(def-class SCHEDULE-EXTENSION-TIME-RANGE-OPERATOR-BODY (lambda-expression) ?x
  "A basic schedule extension operator body is a unary function which takes an
  unassigned job, say ?j and produces a result, ?z, which belongs to the
  resource. ?z is taken as the new resource of ?j in the successor
  schedule state"
  :no-op (:constraint (and (nth-domain ?x 1 job)
    (nth-domain ?x 2 ?sc)
    (=> (= ?z (call ?x ?j))
      (and (has-time-range ?j ?jtr)
        (job-time-range ?z)))))))

(def-class SCHEDULE-EXTENSION-ACTIVITY-OPERATOR (schedule-operator)
  ((applicable-to-jobs :default-value '(setofall ?x (job ?x))
    :type function-expression)
   (has-precondition :default-value (kappa (?schedule-task) (true))
    :type relation-expression)
   (has-body :type schedule-extension-activity-operator-body)))

(def-class SCHEDULE-EXTENSION-ACTIVITY-OPERATOR-BODY (lambda-expression) ?x
  :no-op (:constraint (and (nth-domain ?x 1 job)
    (nth-domain ?x 2 ?sc)
    (=> (= ?z (call ?x ?j))
      (and (has-activities ?j ?list)
        (member ?z ?list)))))))

;;;;;;

(def-class MULTIPLE-SCHEDULE-EXTENSION-RESOURCE-OPERATOR
  (SCHEDULE-EXTENSION-RESOURCE-OPERATOR multiple-operator)
  ((has-body :type multiple-schedule-extension-resource-operator-body)))

(def-class MULTIPLE-SCHEDULE-EXTENSION-RESOURCE-OPERATOR-BODY
  (lambda-expression) ?x
  "A multiple schedule extension operator body is a binary function which takes a job, say
  ?j, and a list of resources, say ?resources, and produces a result, ?z, which belongs to
  the resource range of ?j but is not a member of the list resources ?z is taken as the new
  resource of ?j in the successor schedule state."
  :no-op (:constraint (and (nth-domain ?x 1 job)
    (nth-domain ?x 2 ?y)
    (=> (= ?z (call ?x ?j ?resources))
      (and (requires-resource ?j ?resource)
        (forall ?r (=> (member ?r ?resources)
          (member ?r ?resource)))
        (member ?z ?resource)
        (not (member ?z ?resources)))))))

(def-class MULTIPLE-SCHEDULE-EXTENSION-RESOURCE-TYPE-OPERATOR
  (SCHEDULE-EXTENSION-RESOURCE-TYPE-OPERATOR multiple-operator)
  ((has-body :type multiple-schedule-extension-resource-type-operator-body)))

(def-class MULTIPLE-SCHEDULE-EXTENSION-RESOURCE-TYPE-OPERATOR-BODY
  (lambda-expression) ?x
  "A multiple schedule extension operator body is a binary function which takes a job, say
  ?j, and a list of resource-types, say ?resource-types, and produces a result, ?z, which
  belongs to the resource-type of ?j but is not a member of the list ?resource-types. ?z is
  taken as the new resource-type of ?j in the successor schedule state."
  :no-op (:constraint (and (nth-domain ?x 1 job)
    (nth-domain ?x 2 ?y)
    (=> (= ?z (call ?x ?j ?resource-types))
      (and (requires-resource-type ?j ?resource-type)
        (forall ?rtype (=> (member
          ?r ?resource-types)
          (member
            ?rtype
            ?resource-type)))
        (member ?z ?resource-type)
        (not (member ?z ?resource-types)))))))

```

```

(def-class MULTIPLE-SCHEDULE-EXTENSION-TIME-RANGE-OPERATOR
  (SCHEDULE-EXTENSION-TIME-RANGE-OPERATOR multiple-operator)
  ((has-body :type multiple-schedule-extension-time-range-operator-body)))

(def-class MULTIPLE-SCHEDULE-EXTENSION-TIME-RANGE-OPERATOR-BODY
  (lambda-expression) ?x
  :no-op (:constraint (and (nth-domain ?x 1 job)
                           (nth-domain ?x 2 ?y)
                           (=> (= ?z (call ?x ?j ?job-time-ranges))
                                (and (has-time-range ?j ?job-time-range)
                                     (forall ?jtr (=> (member
                                                         ?jtr ?job-time-ranges)
                                                         (member
                                                          ?jtr ?job-time-range))))
                                (member ?z ?job-time-range)
                                (not (member ?z ?job-time-ranges)))))))

(def-class MULTIPLE-SCHEDULE-EXTENSION-ACTIVITY-OPERATOR
  (schedule-extension-activity-operator multiple-operator)
  ((has-body :type multiple-schedule-extension-activity-operator-body)))

(def-class MULTIPLE-SCHEDULE-EXTENSION-ACTIVITY-OPERATOR-BODY
  (lambda-expression) ?x
  :no-op (:constraint (and (nth-domain ?x 1 job)
                           (nth-domain ?x 2 ?y)
                           (=> (= ?z (call ?x ?j ?activities))
                                (and (has-activities ?j ?list)
                                     (forall ?a (=> (member ?a ?activities)
                                                         (member ?a ?list)))
                                     (member ?z ?list)
                                     (not (member ?z ?activities)))))))

(def-relation PRECONDITION-HOLDS (?op ?sc)
  :constraint (and (schedule-operator ?op)
                  (schedule-model ?sc))
  :iff-def (and (has-precondition ?cp ?exp)
                (holds ?exp ?sc)))

;;;;;;;;;;;;;
;Job dependency network;
;;;;;;;;;;;;;

(def-relation JOB-DEPENDS-ON (?j1 ?j2)
  "This relation states that an assignment of one job, j1, depend on other job, j2."
  :constraint (and (job ?j1)
                  (job ?j2)))

(def-relation JOB-AFFECTS (?j1 ?j2)
  "This relation is an inverse of the relation job-depends-on."
  :constraint (and (job ?j1)
                  (job ?j2))
  :iff-def (job-depends-on ?j2 ?j1))

(def-relation JOB-ASSIGNABLE (?j ?sc)
  "A job is assignable if it is an unassigned job in a schedule and all other jobs that
  depend on it are already assigned."
  :iff-def (and (job ?j)
                (schedule-model ?sc)
                (= ?l (setofall ?x (job-depends-on ?j ?x)))
                (every ?l (kappa (?x)
                                   (assigned-job ?x ?sc)))))

(def-function ALL-ASSIGNABLE-JOBS (?js ?sc)
  "This function retrieves all the unassigned jobs while constructing a schedule."
  :body (setofall ?x (and (member ?x ?js)
                          (unassigned-job ?x ?sc)
                          (job-assignable ?x ?sc))))

(def-function RELEVANT-OPERATORS (?j)
  "This function retrieves all the relevant operators that are necessary to assign a job."
  :constraint (job ?j)
  :body (setofall ?op (and (schedule-operator ?op)
                           (member ?j
                                    (the ?l (applicable-to-jobs ?op ?l))))))

(def-relation APPLICABLE-TO-JOBS (?x ?l)
  "A relation which associates an object such as a constraint or a schedule operator to a
  set of jobs to which the object is 'applicable'"
  :constraint (and (set ?l)
                  (every ?l job)))

```

```

(def-function RELEVANT-CONSTRAINTS (?j)
  :constraint (job ?j)
  :body (setofall ?c (and (psm-constraint ?c)
                           (member ?j
                                     (the ?l (applicable-to-jobs ?c ?l)))))))

(def-class PSM-CONSTRAINT (constraint)
  ((applicable-to-jobs :type function-expression
                       :documentation "An expression which returns the set
                                     of jobs to which this constraint
                                     is applicable")

   (has-expression :cardinality 1
                   :type unary-relation)
   (has-precondition :default-value '(kappa (?j ?schedule-task) (true))
                     :type kappa-expression
                     :documentation "This is an expression which can be used to determine
                                     whether a constraint makes sense for a given job
                                     assignment"))))

(def-class PSM-HARD-CONSTRAINT (PSM-CONSTRAINT))

(def-class PSM-REQUIREMENT (requirement)
  ((applicable-to-jobs :type function-expression
                       :documentation "An expression which returns the set of jobs
                                     to which this requirement is applicable")

   (has-expression :cardinality 1
                   :type unary-relation)
   (has-precondition :default-value '(kappa (?j ?schedule-task) (true))
                     :type kappa-expression
                     :documentation "This is an expression which can be used to
                                     determine whether a requirement makes sense
                                     for a given job assignment"))))

(def-function COLLECT-HARD-CONSTRAINT-VIOLATIONS (?s ?task)
  "Takes a state and a scheduling task and returns
  the set of task hard-constraints which are violated by the
  schedule associated with the state"
  :constraint (and (schedule-state ?s)
                   (scheduling-task ?task))
  :body (setofall ?hc (and (has-schedule-model ?s ?schedule-task)
                           (member ?hc (the ?l (has-hard-constraints
                                                  ?task ?l)))
                           (schedule-violates-constraint
                            ?schedule-task ?constraints)
                           (list ?constraints)
                           (every ?constraints constraint)))))

(def-class PSM-SPECIFIC-JOB () ?j
  "A job is an entity that can be assigned to the resource and has a list of activities."
  ((has-activities :type list
                  :documentation "Each job can have list of activities
                                in order to accomplish the job.")

   (has-activity-type :type activity-type
                     :documentation "It specialises an activity in
                                     more specific types.")
   (requires-resource :type resource :min-cardinality 1
                     :documentation "It says that each job require resources
                                     on which it can be assigned.")
   (requires-resource-type :type resource-type :min-cardinality 1)
   (has-time-range :type job-time-range :max-cardinality 1
                   :documentation "It represents the time range of a
                                   job, within which job must finish.")
   (has-due-date :type calendar-date :max-cardinality 1
                 :documentation "It represents the calendar date for
                                   each job by which it has to dispatch.")
   (has-load :type integer :default-value 1
             :documentation "It represents the number of resources
                               requires by each job.")
   (job-depends-on :type job)
   (job-affects :type job)
   (precedes-job :type job)
   :iff-def (exists ?task (and (scheduling-task ?task)
                                (member ?j (role-value ?task has-jobs)))))

(def-relation POSSIBLE-RESOURCES-FOR-JOB (?j ?r)
  :constraint (job ?j))

(def-relation POSSIBLE-RESOURCE-TYPES-FOR-JOB (?j ?rtype)
  :constraint (job ?j))

(def-relation POSSIBLE-TIME-RANGES-FOR-JOB (?j ?jtr)
  :constraint (job ?j))

```

```
(def-relation POSSIBLE-ACTIVITIES-FOR-JOB (?j ?a)
:constraint (job ?j))
```

```
(def-relation job-precedence-relation (?j ?j1)
:constraint (job ?j))
;;;;;;;;;
```

```

;;;;;;;;;;;;;
;The Generic Model of Scheduling Problem Solving;
;;;;;;;;;;;;;

(def-class PROBLEM-SOLVING-METHOD-FOR-SCHEDULING (problem-solving-method)
  :own-slots ((tackles-task-type scheduling-task)))

(def-class GENERIC-PSM-FOR-SCHEDULING
  (problem-solving-method-for-scheduling decomposition-method)
  ((has-input-role :value has-schedule-operators)
   (has-output-role :value has-solution-state)
   (has-solution-state :type schedule-state)
   (has-schedule-operators :type schedule-operator)
   (has-output-mapping :value '(lambda (?psm ?state)
                                   (the ?sc (has-schedule-model ?state ?sc))))
   (has-body :value '(lambda (?psm)
                        (in-environment
                         ((?s . (achieve-generic-subtask
                                  ?psm gen-schedule-control
                                  has-current-scheduling-task
                                  (the ?task (tackles-task ?psm ?task)))))
                         (if (schedule-state ?s)
                             ?s))))))
  :own-slots ((has-generic-subtasks '(gen-schedule-control))))

(def-class GEN-SCHEDULE-CONTROL (composite-task)
  ((has-input-role :value has-schedule-operators
                  :value has-current-scheduling-task)
   (has-output-role :value has-solution-state)
   (has-solution-state :type schedule-state)
   (has-schedule-operators :type schedule-operator)
   (has-current-scheduling-task :type scheduling-task)
   (has-body :value '(lambda (?psm)
                        (in-environment
                         ((?schedule-space . (achieve-generic-subtask
                                                ?psm generate-schedule-space
                                                has-current-scheduling-task
                                                (role-value
                                                 ?psm
                                                 has-current-scheduling-task))))
                         (REPEAT
                          (in-environment
                           ((?state . (achieve-generic-subtask
                                        ?psm choose-schedule-state
                                        has-schedule-space ?schedule-space)))
                           (if (= ?state :nothing)
                               (return :nothing)
                               (if (achieved (the-current-method) ?state)
                                   (return ?state)
                                   (do
                                    (achieve-generic-subtask
                                     ?psm schedule-from-state
                                     has-schedule-state ?state
                                     has-schedule-space
                                     ?schedule-space))))))))))
   :own-slots ((has-generic-subtasks '(generate-schedule-space
                                       choose-schedule-state
                                       schedule-from-state))))

(def-class GENERATE-SCHEDULE-SPACE (composite-task) ?psm
  ((has-input-role :value has-current-scheduling-task)
   (has-output-role :value has-schedule-space)
   (has-control-role :value has-schedule-model)
   (has-current-scheduling-task :type scheduling-task)
   (has-schedule-space :type schedule-space)
   (has-body :value (lambda (?psm)
                       (in-environment
                        ((?name . (new-symbol 'schedule-space)))
                        (tell (schedule-space
                               ?name has-states nil
                               associated-with-task
                               (role-value
                                ?psm has-current-scheduling-task)))
                        (achieve-generic-subtask
                         ?psm
                         new-schedule-state
                         has-schedule-model nil
                         has-schedule-space ?name)
                         ?name))))
   :own-slots ((has-generic-subtasks '(new-schedule-state))))

```

```

(def-relation STATE-FULLY-EXPANDED (?state)
  :iff-def (and (= ?record (the-state-search-control-record ?state))
    (has-schedule-foci ?record nil)
    (has-schedule-operators ?record nil)))

(def-function SCHEDULE-SPACE-STATE (?space)
  :constraint (schedule-space ?space)
  :body (the ?states (has-states ?space ?states)))

(def-class CHOOSE-SCHEDULE-STATE (goal-specification-task) ?task
  ((has-input-role :value has-schedule-space)
   (has-output-role :value has-schedule-state)
   (has-goal-expression :value (kappa (?task)
    (exists ?s (and (schedule-state ?s)
      (has-schedule-state ?task ?s))))))
  (has-schedule-space :type schedule-space)
  (has-schedule-state :type schedule-state)))

(def-class CONSISTENT-MAXIMAL-CHEAPEST-STATE-SELECTION (primitive-method)
  ((has-body :value (lambda (?psm)
    (in-environment
      ((?cost-algebra . (role-value ?psm has-cost-algebra))
       (?cost-rel . (third ?cost-algebra))
       (?space . (role-value ?psm has-schedule-space))
       (?states . (schedule-space-states ?space)))
      (first
        (filter-cheapest-states
         (filter-maximal-states
          (filter-feasible-consistent-states ?states)
          ?cost-rel)))))))
   :own-slots ((tackles-task-type choose-schedule-state)))

(def-class CONSISTENT-MAXIMAL-STATE-SELECTION (primitive-method)
  ((has-body :value (lambda (?psm)
    (in-environment
      ((?cost-algebra . (role-value ?psm has-cost-algebra))
       (?cost-rel . (third ?cost-algebra))
       (?space . (role-value ?psm has-schedule-space))
       (?states . (schedule-space-states ?space)))
      (first
        (filter-maximal-states
         (filter-feasible-consistent-states ?states)))))))
   :own-slots ((tackles-task-type choose-schedule-state)))

(def-class CONSISTENT-CHEAPEST-MAXIMAL-STATE-SELECTION (primitive-method)
  ((has-body :value (lambda (?psm)
    (in-environment
      ((?cost-algebra . (role-value ?psm has-cost-algebra))
       (?cost-rel . (third ?cost-algebra))
       (?space . (role-value ?psm has-schedule-space))
       (?states . (schedule-space-states ?space)))
      (first
        (filter-maximal-states
         (filter-cheapest-states
          (filter-feasible-consistent-states ?states)
          ?cost-rel)))))))
   :own-slots ((tackles-task-type choose-schedule-state)))

(def-function FILTER-CHEAPEST-STATES (?states ?cost-order-rel)
  :body (setofall ?state (and (member ?state ?states)
    (state-cost ?state ?cost)
    (not (exists ?state2
      (and (member ?state2 ?states)
        (state-cost ?state2 ?cost2)
        (holds ?cost-order-rel
         ?cost2 ?cost)))))))

(def-function FILTER-MAXIMAL-STATES (?states)
  :body (setofall ?state (and (member ?state ?states)
    (has-schedule-model ?state ?sc)
    (= ?l1 (length ?sc))
    (not (exists ?state2
      (and (member ?state2 ?states)
        (has-schedule-model
         ?state2 ?sc2)
        (= ?l2 (length ?sc2))
        (> ?l2 ?l1)))))))

```

```

(def-function FILTER-FEASIBLE-CONSISTENT-STATES (?states)
:body (setofall ?state (and (member ?state ?states)
                             (not (deadend-state ?state))
                             (not (constraint-violations ?state))
                             (not (requirement-violations ?state)))))

(def-class NEW-SCHEDULE-STATE (composite-task) ?psm
  (has-input-role :value has-schedule-model
                  :value has-schedule-space)
  (has-output-role :value has-schedule-state)
  (has-schedule-space :type schedule-space)
  (has-schedule-state :type schedule-state)
  (has-schedule-model :type schedule-model)
  (has-body :value (lambda (?psm)
                     (in-environment
                      ((?sc . (the ?sc2 (has-schedule-model ?psm ?sc2)))
                       (?schedule-space . (role-value
                                           ?psm has-schedule-space))
                       (?name . (new-symbol 'schedule-state)))
                      (tell (schedule-state ?name
                                           has-schedule-model ?sc))
                      (append-slot-value ?schedule-space has-states ?name)
                      (achieve-generic-subtask
                       ?psm apply-downstream-consistency-enforcement-mechanism
                       has-schedule-state ?name)
                      (achieve-generic-subtask ?psm evaluate-schedule-state
                                           has-schedule-state ?name)
                      ?name))))
  :own-slots ((has-generic-subtasks
                ' (apply-downstream-consistency-enforcement-mechanism
                  evaluate-schedule-state))))
;;;;;;;;;;

(def-class APPLY-DOWNSTREAM-CONSISTENCY-ENFORCEMENT-MECHANISM
  (goal-specification-task)
  "This is a simple heuristics which propagates the earliest start time of job-1 such that
all the jobs that has later start time than job-1 precedes the job-1. The complexity of
this heuristics is linear and in the absence of the Resource-capacity it ensures backtrack-
free search."
  ((has-input-role :value has-schedule-state)
   (has-output-role
    :value has-schedule-state-with-enforced-downstream-consistency)
   (has-schedule-state-with-enforced-downstream-consistency :type schedule-state)
   (has-schedule-state :type schedule-state)
   (has-goal-expression
    :value (kappa (?task ?state)
                  (and (has-schedule-state-with-enforced-downstream-consistency
                        ?task ?state)
                      (schedule-state ?state)))))

(def-class APPLICATION-OF-DOWNSTREAM-CONSISTENCY-MECHANISM (primitive-method)
  ((has-input-role :value has-schedule-state)
   (has-schedule-state :type schedule-state)
   (has-body :value
    (lambda (?psm)
      (in-environment
       ((?state . (role-value ?psm has-schedule-state))
        (?schedule-model . (the ?schedule-model
                               (has-schedule-model
                                ?state ?schedule-model)))
        (?jobs . (role-value ?psm has-jobs)))
       (downstream-consistency-enforced-schedule-state
        ?state ?jobs)))))
  :own-slots ((tackles-task-type
                'apply-downstream-consistency-enforcement-mechanism)))

(def-function DOWNSTREAM-CONSISTENCY-ENFORCED-SCHEDULE-STATE (?state ?jobs)
:constraint (and (list ?jobs)
                 (every ?jobs job)
                 (schedule-state ?state))
:body (setofall ?job
  (and (has-schedule-model ?state ?schedule-model)
        (has-jobs ?schedule-model ?jobs)
        (member ?job ?jobs)
        (has-time-range ?job ?jtr)
        (= (start-time-of-a-job ?job ?jtr) ?est)
        (exists ?job2 (and (member ?job2 ?jobs)
                           (has-time-range ?job2 ?jtr2)
                           (= (start-time-of-a-job ?job2 ?jtr2)
                              ?est2)
                           (job-start-time-earlier-than
                            ?est ?est2)
                           (job-precedes ?job ?job2)))))
;;;;;;;;;;

```

```

(def-relation DEADEND-STATE (?state)
  "A deadend state is the one from which solution cannot be derived."
  :constraint (schedule-state ?state))

(def-relation STATE-COMPLETE (?state ?jobs)
  "A state is complete is a schedule associated with a state is a complete one."
  :iff-def (and (has-schedule-model ?state ?schedule-model)
    (schedule-minimally-complete ?schedule-model ?jobs)))

(def-relation SOLUTION-STATE (?state)
  "A state is a solution state if a schedule associated with this state is a complete one,
  i.e. all the jobs are assigned to the resources and have the correct time ranges."
  :constraint (state-complete ?state))

(def-relation CONSTRAINT-VIOLATIONS (?state ?cs)
  :constraint (and (schedule-state ?state)
    (list ?cs)
    (every ?cs constraint)))

(def-relation REQUIREMENT-VIOLATIONS (?state ?requirements)
  :constraint (and (schedule-state ?state)
    (list ?requirements)
    (every ?requirements requirement)))

(def-relation STATE-FEASIBLE (?state)
  "A state is feasible if it does not violate any requirements imposed on a schedule
  associated with it."
  :iff-def (and (has-schedule-model ?state ?schedule-model)
    (not (requirement-violations ?state ?requirements))))

(def-relation STATE-COST (?state ?cost)
  :constraint (and (schedule-state ?state)
    (cost ?cost)))

(def-class EVALUATE-SCHEDULE-STATE (composite-task) ?task
  ((has-input-role :value has-schedule-state)
   (has-schedule-state :type schedule-state)
   (has-body :value (lambda (?task)
     (in-environment
      ((?state . (role-value ?task has-schedule-state)))
      (achieve-generic-subtask
       ?task evaluate-hard-consistency
       has-schedule-state ?state)
      (achieve-generic-subtask ?task evaluate-completeness
        has-schedule-state ?state)
      (achieve-generic-subtask ?task evaluate-cost
        has-schedule-state ?state)
      (achieve-generic-subtask
       ?task evaluate-current-job-consistency
       has-schedule-state ?state)
      (achieve-generic-subtask
       ?task evaluate-future-job-consistency
       has-schedule-state ?state)
      (achieve-generic-subtask ?task evaluate-feasibility
        has-schedule-state ?state)))))))

(def-class EVALUATE-COST (goal-specification-task) ?task
  ((has-input-role :value has-schedule-state)
   (has-output-role :value has-cost)
   (has-schedule-state :type schedule-state)
   (has-cost :type cost)
   (has-goal-expression :value (kappa (?task ?cost)
     (and (cost ?cost)
      (has-cost ?task ?cost))))))

(def-class DEFAULT-COST-EVALUATION (primitive-method) ?psm
  ((has-body :value (lambda (?psm)
    (in-environment
      ((?state . (role-value ?psm has-schedule-state))
       (?schedule-model . (the ?sc (has-schedule-model
        ?state ?sc))))
      (?cost-fun . (role-value ?psm has-cost-function))
      (?cost . (call ?cost-fun ?schedule-model)))
    (do
      (tell (state-cost ?state ?cost))
      ?cost))))))
  :own-slots ((tackles-task-type evaluate-cost)))

```

```

(def-class EVALUATE-HARD-CONSISTENCY (primitive-task) ?task
  ((has-input-role :value has-schedule-state)
   (has-schedule-state :type schedule-state)
   (has-body :value (lambda (?task)
                       (in-environment
                        ((?state . (role-value ?task has-schedule-state))
                         (?schedule-model . (the ?sc (has-schedule-model
                                                         ?state ?sc)))
                         (?hard-constraints . (role-value
                                                ?task has-hard-constraints))
                         (?hcv . (setofall ?hc (and (member ?hc ?hard-constraints)
                                                    (every ?hard-constraints
                                                         hard-constraint)
                                                         (schedule-violates-constraint
                                                          ?schedule-model ?constraints)
                                                         (every ?constraints
                                                             constraint))))))
                        (if (not (null ?hcv))
                            (tell (constraint-violations ?state ?hcv)))
                          ?hcv))))))

(def-class EVALUATE-FEASIBILITY (primitive-task) ?task
  ((has-input-role :value has-schedule-state)
   (has-schedule-state :type schedule-state)
   (has-body :value (lambda (?task)
                       (in-environment
                        ((?state . (role-value ?task has-schedule-state))
                         (?schedule-model . (the ?sc (has-schedule-model
                                                         ?state ?sc)))
                         (?requirements . (role-value ?task has-requirements))
                         (?reqv . (setofall ?req
                                             (and (member ?req ?requirements)
                                                  (every ?requirements requirement)
                                                  (schedule-violates-requirement
                                                   ?schedule-model ?requirements))))))
                        (if (not (null ?reqv))
                            (tell (requirement-violations ?state ?reqv)))
                          ?reqv))))))

(def-class EVALUATE-COMPLETENESS (primitive-task) ?task
  ((has-input-role :value has-schedule-state)
   (has-schedule-state :type schedule-state)
   (has-body :value (lambda (?task)
                       (in-environment
                        ((?state . (role-value ?task has-schedule-state))
                         (?schedule-model . (the ?sc (has-schedule-model
                                                         ?state ?sc)))
                         (?jobs . (role-value ?task has-jobs)))
                        (if (schedule-minimally-complete ?schedule-model ?jobs)
                            (tell (state-complete ?state))))))))))

(def-class EVALUATE-CURRENT-JOB-CONSISTENCY (primitive-task)
  "This method checks the consistency of the jobs by comparing the compatibility of resource
  requirement between assigned jobs and yet-to-be assigned jobs. If the resource requirements
  of these jobs are not consistent then it gives all those inconsistent jobs within a
  schedule-state."
  ((has-input-role :value has-schedule-state)
   (has-schedule-state :type schedule-state)
   (has-body :value
    (lambda (?task)
      (in-environment
       ((?state . (role-value ?task has-schedule-state))
        (?schedule-model . (the ?schedule-model (has-schedule-model
                                                    ?state ?schedule-model)))
        (?jobs . (role-value ?task has-jobs)))
       (if (job-consistency-in-schedule-state ?state ?jobs)
           (tell (schedule-state-consistent ?state))))))))

```

```

(def-relation JOB-CONSISTENCY-IN-SCHEDULE-STATE (?state ?jobs)
  "This relation says that the resource assigned to any of the jobs (i.e. assigned-job) is
  not equal to the possible resource requirement of any other jobs (i.e. assignable job) in a
  schedule. And these jobs are dependent on each other."
  :constraint (and (schedule-state ?state has-schedule-model ?schedule-model)
    (has-jobs ?schedule-model ?jobs)
    (every ?jobs job))
  :iff-def (or (exists ?j1 (and
    (member ?j1 ?jobs)
    (assigned-job ?j1 ?schedule-model)
    (= (resource-assigned-to-a-job
      ?j1 ?schedule-model) ?r1)
    (not (exists ?j2 (and (member ?j2 ?jobs)
      (unassigned-job
        ?j2 ?schedule-model)
      (= (expected-resources-for-job
        ?j2 ?schedule-model) ?r2)
      (= ?r1 ?r2))))))
    (exists ?j1 (and (member ?j1 ?jobs)
      (assigned-job ?j1 ?schedule-model)
      (= (resource-assigned-to-a-job
        ?j1 ?schedule-model) ?r1)
      (not (exists ?j2 (and (member ?j2 ?jobs)
        (assigned-job
          ?j2 ?schedule-model)
        (= (resource-assigned-to-a-job
          ?j2 ?schedule-model) ?r2)
        (= ?r1 ?r2))))))))))

(def-function EXPECTED-RESOURCES-FOR-JOB (?job ?sc) -> ?r
  :constraint (and (job ?job)
    (schedule-model ?sc)
    (resource ?r)
    (requires-resource ?job ?r))
  :body (setofall ?r (assigned-to-resource ?job ?r ?sc)))

(def-class EVALUATE-FUTURE-JOB-CONSISTENCY (primitive-task)
  "This method checks the consistency of all yet-to-be assigned jobs (i.e., future jobs) in
  terms of the compatibility between their resource requirements. If the resource requirement
  conflicts with each other then it returns all those inconsistent jobs within a schedule-
  state."
  (has-input-role :value has-schedule-state)
  (has-schedule-state :type schedule-state)
  (has-body :value
    '(lambda (?task)
      (in-environment
        ((?state . (role-value ?task has-schedule-state))
         (?schedule-model . (the ?schedule-model (has-schedule-model
           ?state ?schedule-model))))
        (?jobs . (role-value ?task has-jobs)))
      (if (future-job-consistency-in-schedule-state ?state ?jobs)
        (tell (schedule-state-consistent ?state)))))))

(def-relation FUTURE-JOB-CONSISTENCY-IN-SCHEDULE-STATE (?state ?jobs)
  "This relation says that the resource requirement of assignable job j1 is not equal to
  another assignable job j2 in a schedule. And both these jobs are dependent on each other
  and are affected by each other."
  :constraint (and (schedule-state ?state has-schedule-model ?schedule-model)
    (has-jobs ?schedule-model ?jobs)
    (every ?jobs job))
  :iff-def (exists ?j1 (and (member ?j1 ?jobs)
    (unassigned-job ?j1 ?schedule-model)
    (= (expected-resources-for-job ?j1 ?schedule-model)
      ?r1)
    (not (exists ?j2
      (and (member ?j2 ?jobs)
        (unassigned-job
          ?j2 ?schedule-model)
        (= (expected-resources-for-job
          ?j2 ?schedule-model) ?r2)
        (= ?r1 ?r2)))))))

;;;;;;;;;;

```

```

(def-class SCHEDULE-FROM-STATE (goal-specification-task) ?task
  ((has-input-role :value has-schedule-state
                   :value has-schedule-space)
   (has-output-role :value has-output-state)
   (has-output-state :type schedule-state)
   (has-schedule-state :type schedule-state)
   (has-schedule-space :type schedule-space)
   (has-goal-expression :value (kappa (?task ?s)
                                       (schedule-state ?s))))
  :constraint (and (has-schedule-state ?task ?s)
                   (has-schedule-model ?s ?sc)
                   (= ?scheduling-problem (role-value
                                           ?task has-current-scheduling-task))
                   (not (achieved ?scheduling-problem ?sc))))

(def-class EXPAND-INCOMPLETE-STATE (decomposition-method)
  ((has-input-role :value has-schedule-state)
   (has-output-role :value generates-schedule-state)
   (has-schedule-state :type schedule-state)
   (generates-schedule-state :type schedule-state)
   (has-goal-expression :value (kappa (?task ?s)
                                       (schedule-extends
                                        (the ?sc (has-schedule-model ?s ?sc))
                                        (the ?sc (has-schedule-model
                                                  (role-value
                                                    ?task has-schedule-state)
                                                    ?sc)))))))
  (has-body :value '(lambda (?psm)
                      (in-environment
                       ((?state . (role-value ?psm has-schedule-state))
                        (?schedule-model . (the ?sc (has-schedule-model
                                                       ?state ?sc)))
                        (?hard-constraints . (role-value
                                              ?psm has-hard-constraints))
                        (?requirements . (role-value ?psm has-requirements))
                        (?jobs . (role-value ?psm has-jobs)))
                       (if (deadend-state ?state)
                           :nothing
                           (if (constraint-violations ?state ?constraints)
                               (tell (deadend-state ?state))
                               (if (deadend-state ?state)
                                   :nothing
                                   (if (requirement-violations ?state ?requirements)
                                       (tell (deadend-state ?state))
                                       (if (solution-state ?state)
                                           (return ?state)
                                           (do
                                            (achieve-generic-subtask
                                             ?psm
                                             generate-new-state-successor
                                             has-schedule-state ?state
                                             has-schedule-context :extend))))))))))
  :own-slots ((tackles-task-type schedule-from-state)
              (has-generic-subtasks generate-new-state-successor)))

```

```

(def-class GENERATE-NEW-STATE-SUCCESSOR (composite-task)
  ((has-input-role :value has-schedule-state
                   :value has-schedule-context)
   (has-output-role :value generates-schedule-state)
   (has-schedule-context :type schedule-context)
   (has-schedule-state :type schedule-state)
   (generates-schedule-state :type schedule-state)
   (has-body :value (lambda (?task)
                      (in-environment
                       ((?state . (role-value ?task has-schedule-state))
                        (?js . (role-value ?task has-jobs))
                        (?context . (role-value ?task has-schedule-context))))
                    (if (search-control-record
                        ?record has-schedule-state ?state)
                        (in-environment
                         ((?result . (achieve-generic-subtask
                                      ?task
                                      resume-state has-schedule-state ?state
                                      has-schedule-context
                                      ?context)))
                         (if (schedule-state ?result)
                             ?result
                             (achieve-generic-subtask
                              ?task propose-schedule-from-context
                              has-schedule-state ?state
                              has-schedule-context ?context)))
                        (in-environment
                         ((?foci . (achieve-generic-subtask
                                    ?task collect-state-foci
                                    has-schedule-state ?state
                                    has-schedule-context ?context)))
                         (new-search-control-record ?state ?foci)
                         (achieve-generic-subtask
                          ?task propose-schedule-from-context
                          has-schedule-state ?state
                          has-schedule-context ?context)))))))
  :own-slots ((has-generic-subtasks '(resume-state
                                       propose-schedule-from-context
                                       collect-state-foci))))

(def-class COLLECT-STATE-FOCI (goal-specification-task) ?task
  ((has-input-role :value has-schedule-context
                   :value has-schedule-state)
   (has-output-role :value has-schedule-foci)
   (has-schedule-foci :type list)
   (has-schedule-state :type schedule-state)
   (has-schedule-context :type schedule-context)))

(def-class COLLECT-ASSIGNABLE-JOBS (primitive-method)
  ((has-body :value (lambda (?psm)
                     (all-assignable-jobs
                      (role-value ?psm has-jobs)
                      (the ?sc (has-schedule-model
                               (role-value ?psm has-schedule-state)
                               ?sc))))))
  :own-slots ((tackles-task-type collect-state-foci)))

```

```

(def-class PROPOSE-SCHEDULE-FROM-CONTEXT (composite-task) ?task
  ((has-input-role :value has-schedule-state
                   :value has-schedule-context)
   (has-output-role :value generates-schedule-state)
   (has-control-role :value has-schedule-foci
                     :value has-search-control-record)
   (has-schedule-context :type schedule-context)
   (has-schedule-state :type schedule-state)
   (generates-schedule-state :type schedule-state)
   (has-body :value (lambda (?task)
                       (repeat
                        (in-environment
                         ((?state . (role-value ?task has-schedule-state))
                          (?record . (the-slot-value ?record 'has-schedule-foci))
                          (?foci . (the-slot-value ?record 'has-schedule-foci))
                          (?sub . (instantiate-generic-subtask
                                   ?task select-schedule-focus
                                   has-schedule-foci ?foci))
                          (?focus . (solve-task ?sub)))
                         (if (achieved ?sub ?focus)
                             (do
                              (achieve-generic-subtask
                               ?task
                               amend-search-control-record-on-focus-selection
                               has-search-control-record ?record
                               has-schedule-focus ?focus)
                              (in-environment
                               ((?ops . (achieve-generic-subtask
                                           ?task collect-focus-operators
                                           has-schedule-focus ?focus))
                                (?sorted-ops . (achieve-generic-subtask
                                                  ?task sort-schedule-operators
                                                  has-schedule-operators
                                                  ?ops)))
                               (if (null ?sorted-ops)
                                   (achieve-generic-subtask
                                    ?task
                                    amend-search-control-record-on-focus-failure
                                    has-search-control-record ?record
                                    has-schedule-focus ?focus)
                                   (do
                                    (set-slot-value ?record
                                                       has-schedule-operators
                                                       ?sorted-ops)
                                    (in-environment
                                     ((?value . (achieve-generic-subtask
                                                  ?task
                                                  generate-value-from-focus
                                                  has-schedule-state ?state)))
                                     (if (not (= ?value :nothing))
                                         (in-environment
                                          ((?activity-value . (achieve-generic-subtask
                                                                ?task
                                                                generate-activities-from-
                                                                    has-schedule-state
                                                                    ?state)))
                                         (if (not (= ?activity-value :nothing))
                                             (in-environment
                                              ((?result . (achieve-generic-subtask
                                                            ?task propose-schedule-from-focus
                                                            has-schedule-state
                                                            ?state
                                                            has-schedule-value
                                                            ?value
                                                            has-schedule-activity-value
                                                            ?activity-value)))
                                              (if (schedule-state ?result)
                                                  (return ?result))))))))))
                              (do
                               (tell (deadend-state ?state))
                               (return :nothing)))))))
                             :own-slots ((has-generic-subtasks
                                           ' (select-schedule-focus collect-focus-operators
                                              sort-schedule-operators
                                              amend-search-control-record-on-focus-selection
                                              amend-search-control-record-on-focus-failure
                                              generate-value-from-focus generate-activities-from-focus
                                              propose-schedule-from-focus))))

```

```

(def-class SELECT-SCHEDULE-FOCUS (goal-specification-task) ?task
  ((has-input-role :value has-schedule-foci)
   (has-output-role :value has-schedule-focus)
   (has-schedule-foci :type list)
   (has-schedule-focus :type schedule-focus)
   (has-goal-expression :value (kappa (?task ?focus)
                                       (has-schedule-focus ?task ?focus)))))

(def-class DEFAULT-JOB-SELECTION (primitive-method) ?psm
  ((has-input-role :value has-schedule-focus-order-relation
                   :value has-possible-resources-relation)
   (has-schedule-focus-order-relation :default-value schedule-focus-order)
   (has-possible-resources-relation :default-value possible-resources-for-job)
   (has-body :value (lambda (?psm)
                       (if (= ?foci (role-value ?psm has-schedule-foci))
                           (select-most-preferred-focus
                            (collect-most-restricted-jobs
                             ?foci
                             (role-value ?psm has-possible-resources-relation))
                            (role-value
                             ?psm has-schedule-focus-order-relation))))))
  :own-slots ((tackles-task-type select-schedule-focus)))

(def-function COLLECT-MOST-RESTRICTED-JOBS (?l ?rel)
  :body (in-environment
        ((?quadruples . (sort (map '(lambda (?j)
                                     (list-of
                                      ?j (setofall ?r (holds ?rel ?j ?r))))
                                   ?l)
                              '(kappa (?x ?y)
                                       (< (length (second ?x))
                                           (length (second ?y)))))))
        (map first (filter
                    ?quadruples
                    '(kappa (?quadruple)
                            (= (first ?quadruple)
                               (first (first ?quadruples)))))))

(def-class AMEND-SEARCH-CONTROL-RECORD-ON-FOCUS-SELECTION (goal-specification-task)
  ((has-input-role :value has-search-control-record
                   :value has-schedule-focus)
   (has-schedule-focus :type schedule-focus)
   (has-search-control-record :type search-control-record)))

(def-class DEFAULT-SEARCH-CONTROL-RECORD-ON-FOCUS-SELECTION-UPDATE
  (primitive-method) ?psm
  ((has-body :value (lambda (?psm)
                      (in-environment
                       ((?focus . (role-value ?psm has-schedule-focus))
                        (?record . (role-value
                                   ?psm has-search-control-record)))
                       (set-slot-value ?record has-schedule-foci
                                       (remove ?focus
                                                (the-slot-value
                                                 ?record has-schedule-foci)))
                       (set-slot-value ?record has-schedule-focus ?focus))))
   :own-slots ((tackles-task-type amend-search-control-record-on-focus-selection)))

(def-class AMEND-SEARCH-CONTROL-RECORD-ON-FOCUS-FAILURE
  (goal-specification-task) ?task
  ((has-input-role :value has-search-control-record
                   :value has-schedule-focus)
   (has-schedule-focus :type schedule-focus)
   (has-search-control-record :type search-control-record)))

(def-class DEFAULT-SEARCH-CONTROL-RECORD-ON-FOCUS-FAILURE-UPDATE
  (primitive-method) ?psm
  ((has-body :value (lambda (?psm) :nothing)))
  :own-slots ((tackles-task-type amend-search-control-record-on-focus-failure)))

```

```

(def-class COLLECT-FOCUS-OPERATORS (goal-specification-task) ?task
  ((has-input-role :value has-schedule-focus)
   (has-schedule-focus :type schedule-focus)))

(def-class DEFAULT-OPERATOR-COLLECTION (primitive-method) ?psm
  ((has-body :value (lambda (?psm)
                      (setofall ?op
                                (and (schedule-operator ?op
                                                           applicable-to-jobs ?l)
                                     (member (role-value
                                              ?psm 'has-schedule-focus)
                                              (eval ?l)))))))
   :own-slots ((tackles-task-type collect-focus-operators)))

(def-class SORT-SCHEDULE-OPERATORS (primitive-task) ?task
  ((has-input-role :value has-schedule-operators
                  :value has-operator-order-relation)
   (has-schedule-operators :type list)
   (has-operator-order-relation :default-value schedule-operator-order)
   (has-body :value (lambda (?task)
                      (sort (role-value
                             ?task has-schedule-operators)
                            (role-value ?task has-operator-order-relation))))))

(def-class RESUME-STATE (goal-specification-task) ?task
  ((has-input-role :value has-schedule-state
                  :value has-schedule-context)
   (has-output-role :value has-output-schedule-state)
   (has-schedule-state :type schedule-state)
   (has-schedule-context :type schedule-context)
   (has-output-schedule-state :type schedule-state)
   (has-goal-expression :value (kappa (?task ?s)
                                       (and (schedule-state ?s)
                                             (not (= ?s (role-value
                                                         ?task
                                                         has-schedule-state)))))))

(def-class TRY-DIFFERENT-STATE-OPERATOR (primitive-method) ?psm
  ((has-body :value (lambda (?psm)
                      (achieve-generic-subtask
                       ?psm propose-schedule-from-focus
                       has-schedule-state (role-value
                                           ?psm has-schedule-state))))
   :own-slots ((tackles-task-type resume-state)))

(def-class RETRY-SCHEDULE-STATE-OPERATOR (primitive-method) ?psm
  ((has-body :value (lambda (?psm)
                      (in-environment
                       ((?state . (role-value ?psm has-schedule-state))
                        (?record . (the-state-search-control-record ?state))
                        (?op . (the ?op2 (has-current-operator ?record ?op2))))
                       (if (has-schedule-focus ?record ?focus)
                           (in-environment
                            ((?sub . (instantiate-generic-subtask
                                       ?psm try-schedule-operator
                                       has-schedule-operator ?op
                                       has-schedule-focus ?focus
                                       has-schedule-model
                                       (the-slot-value
                                        ?state 'has-schedule-model)))
                             (?result . (solve-task ?sub3)))
                            (if (achieved ?sub3 ?result)
                                ?result
                                (achieve-generic-subtask
                                 ?psm
                                 propose-schedule-from-focus
                                 has-schedule-state ?state))))))
                      :own-slots ((tackles-task-type resume-state)))

(def-class SEARCH-CONTROL-RECORD ()
  ((has-schedule-state :type schedule-state :cardinality 1)
   (has-schedule-focus :type schedule-focus :cardinality 1)
   (has-current-operator :type schedule-operator :max-cardinality 1)
   (has-schedule-operators :type list :cardinality 1)
   (has-schedule-foci :type list :cardinality 1)))

(def-function THE-STATE-SEARCH-CONTROL-RECORD (?state)
  :body (the ?record (and (search-control-record ?record)
                          (has-schedule-state ?record ?state))))

```

```

(def-procedure NEW-SEARCH-CONTROL-RECORD (?state ?foci)
  :body (tell
    (search-control-record
      (new-symbol 'state-search-control-record)
      has-schedule-state ?state
      has-schedule-foci ?foci)))

(def-class GENERATE-VALUE-FROM-FOCUS (composite-task)
  ((has-input-role :value has-schedule-state)
   (has-output-role :value has-schedule-value)
   (has-control-role :value has-schedule-model
                     :value has-schedule-operator)
   (has-schedule-state :type schedule-state)
   (has-schedule-focus :type schedule-focus)
   (has-schedule-value :type schedule-value)
   (has-goal-expression :value (kappa (?task ?value)
                                       (and (has-schedule-value ?task ?value)
                                             (schedule-value ?value)))))

  (has-body :value (lambda (?task)
    (REPEAT
      (in-environment
        ((?state . (role-value ?task has-schedule-state))
         (?record . (the-state-search-control-record ?state))
         (?focus . (the-slot-value
                     ?record 'has-schedule-focus))
         (?ops . (the-slot-value
                  ?record 'has-schedule-operators))
         (?sub1 . (instantiate-generic-subtask
                   ?task
                   select-resource-operator
                   has-schedule-focus ?focus
                   has-schedule-operators ?ops))
         (?op . (solve-task ?sub1)))
        (set-slot-value ?record has-current-operator ?op)
        (if (achieved ?sub1 ?op)
          (do
            (set-slot-value ?record
                          has-schedule-operators
                          (remove ?op ?ops))
            (in-environment
              ((?sub3 . (instantiate-generic-subtask
                        ?task
                        try-schedule-resource-operator
                        has-schedule-operator ?op
                        has-schedule-focus ?focus
                        has-schedule-model
                        (the-slot-value
                          ?state
                          'has-schedule-model)))
              (?value . (solve-task ?sub3)))
              (if (achieved ?value ?sub3)
                (return ?value)))
            (return :nothing))))))
    :own-slots ((has-generic-subtasks '(select-resource-operator
                                       try-schedule-resource-operator))))

(def-class SELECT-RESOURCE-OPERATOR (goal-specification-task)
  ((has-input-role :value has-schedule-focus
                  :value has-schedule-operators)
   (has-output-role :value has-selected-resource-operator)
   (has-schedule-focus :type schedule-focus)
   (has-schedule-operators :type list)
   (has-selected-resource-operator :type schedule-operator)
   (has-goal-expression :value (kappa (?task ?op)
                                       (and (schedule-operator ?op)
                                             (has-selected-resource-operator
                                              ?task ?op)))))

(def-class DEFAULT-RESOURCE-OPERATOR-SELECTION (primitive-method)
  ((has-body :value (lambda (?psm)
    (first (role-value ?psm
                      'has-schedule-operators))))
   :own-slots ((tackles-task-type select-resource-operator)))

```

```

(def-class TRY-SCHEDULE-RESOURCE-OPERATOR (goal-specification-task)
  ((has-input-role :value has-schedule-operator
                   :value has-schedule-focus
                   :value has-schedule-model)
   (has-output-role :value has-schedule-value)
   (has-schedule-operator :type schedule-operator)
   (has-schedule-focus :type schedule-focus)
   (has-schedule-model :type schedule-model)
   (has-schedule-value :type schedule-value)
   (has-goal-expression :value (kappa (?task ?value)
                                       (and (has-schedule-value ?task ?value)
                                             (schedule-value ?value))))))

(def-class TRY-SCHEDULE-EXTENSION-RESOURCE-OPERATOR (primitive-method)
  ((has-body :value (lambda (?psm)
                      (in-environment
                       ((?sc . (role-value ?psm 'has-schedule-model))
                        (?focus . (role-value ?psm 'has-schedule-focus))
                        (?value . (apply-schedule-extension-resource-operator
                                  ?focus ?sc
                                  (role-value ?psm 'has-schedule-operator))))
                      (if (not (= ?value :nothing))
                          (return ?value))))))
   :own-slots ((tackles-task-type try-schedule-resource-operator)))

(def-function APPLY-SCHEDULE-EXTENSION-RESOURCE-OPERATOR (?j ?sc ?op)
  :constraint (and (job ?j)
                   (schedule-model ?sc)
                   (schedule-extension-resource-operator ?op))
  :body (call (the ?body
                  (has-body ?op ?body)) ?j ?sc))

(def-class GENERATE-ACTIVITIES-FROM-FOCUS (composite-task)
  ((has-input-role :value has-schedule-state)
   (has-output-role :value has-schedule-activity-value)
   (has-control-role :value has-schedule-model
                     :value has-schedule-operator)
   (has-schedule-state :type schedule-state)
   (has-schedule-activity-value :type activity-value)
   (has-schedule-focus :type schedule-focus)
   (has-body :value (lambda (?task)
                      (REPEAT
                       (in-environment
                        ((?state . (role-value ?task has-schedule-state))
                         (?record . (the-state-search-control-record ?state))
                         (?focus . (the-slot-value
                                     ?record 'has-schedule-focus))
                         (?ops . (the-slot-value
                                   ?record 'has-schedule-operators))
                         (?sub5 . (instantiate-generic-subtask
                                   ?task select-activity-operator
                                   has-schedule-focus ?focus
                                   has-schedule-operators ?ops))
                         (?op . (solve-task ?sub5))))
                       (set-slot-value ?record has-current-operator ?op)
                       (if (achieved ?sub5 ?op)
                           (do
                            (set-slot-value ?record
                                              has-schedule-operators
                                              (remove ?op ?ops))
                            (in-environment
                             ((?sub7 . (instantiate-generic-subtask
                                         ?task
                                         try-schedule-activity-operator
                                         has-schedule-operator ?op
                                         has-schedule-focus ?focus
                                         has-schedule-model
                                         (the-slot-value
                                          ?state
                                          'has-schedule-model)))
                              (?activity-value . (solve-task ?sub7)))
                              (if (achieved ?activity-value ?sub7)
                                  (return ?activity-value))))
                           (return :nothing))))))
   :own-slots ((has-generic-subtasks '(select-activity-operator
                                       try-schedule-activity-operator))))

```

```

(def-class SELECT-ACTIVITY-OPERATOR (goal-specification-task)
  ((has-input-role :value has-schedule-focus
                   :value has-schedule-operators)
   (has-output-role :value has-selected-activity-operator)
   (has-schedule-focus :type schedule-focus)
   (has-schedule-operators :type list)
   (has-selected-activity-operator :type schedule-operator)
   (has-goal-expression :value (kappa (?task ?op)
                                       (and (schedule-operator ?op)
                                             (has-selected-activity-operator
                                              ?task ?op))))))

(def-class DEFAULT-ACTIVITY-OPERATOR-SELECTION (primitive-method)
  ((has-body :value (lambda (?psm)
                      (first (role-value ?psm
                                          'has-schedule-operators))))
   :own-slots ((tackles-task-type select-activity-operator)))

(def-class TRY-SCHEDULE-ACTIVITY-OPERATOR (goal-specification-task)
  ((has-input-role :value has-schedule-operator
                   :value has-schedule-focus
                   :value has-schedule-model)
   (has-output-role :value has-schedule-activity-value)
   (has-schedule-operator :type schedule-operator)
   (has-schedule-focus :type schedule-focus)
   (has-schedule-model :type schedule-model)
   (has-schedule-activity-value :type activity-value)
   (has-goal-expression :value (kappa (?task ?activity-value)
                                       (and (has-schedule-activity-value
                                              ?task ?activity-value)
                                            (activity-value ?activity-value))))))

(def-class TRY-SCHEDULE-EXTENSION-ACTIVITY-OPERATOR (primitive-method)
  ((has-body :value (lambda (?psm)
                      (in-environment
                       ((?sc . (role-value ?psm has-schedule-model))
                        (?focus . (role-value ?psm has-schedule-focus))
                        (?activity-value . (apply-schedule-extension-activity-operator
                                           ?focus ?sc
                                           (role-value
                                            ?psm 'has-schedule-operator))))
                       (if (not (= ?activity-value :nothing))
                           (return ?activity-value))))))
   :own-slots ((tackles-task-type try-schedule-activity-operator)))

(def-function APPLY-SCHEDULE-EXTENSION-ACTIVITY-OPERATOR (?j ?sc ?op)
  :constraint (and (job ?j)
                  (schedule-model ?sc)
                  (schedule-extension-activity-operator ?op))
  :body (call (the ?body
                  (has-body ?op ?body)) ?j ?sc))

```

```

(def-class PROPOSE-SCHEDULE-FROM-FOCUS (composite-task)
  ((has-input-role :value has-schedule-state
                   :value has-schedule-value
                   :value has-schedule-activity-value)
   (has-output-role :value has-output-schedule-state)
   (has-control-role :value has-schedule-model
                     :value has-schedule-operator)
   (has-schedule-state :type schedule-state)
   (has-schedule-value :type schedule-value)
   (has-schedule-activity-value :type activity-value)
   (has-output-schedule-state :type schedule-state)
   (has-body :value (lambda (?task)
                       (repeat
                        (in-environment
                         ((?state . (role-value ?task has-schedule-state))
                          (?record . (the-state-search-control-record ?state))
                          (?focus . (the-slot-value
                                      ?record 'has-schedule-focus))
                          (?ops . (the-slot-value
                                   ?record 'has-schedule-operators))
                          (?value . (role-value ?task has-schedule-value))
                          (?activity-value . (role-value
                                              ?task
                                              has-schedule-activity-value))
                          (?sub . (instantiate-generic-subtask
                                   ?task
                                   select-schedule-operator
                                   has-schedule-focus ?focus
                                   has-schedule-operators ?ops))
                          (?op . (solve-task ?sub)))
                         (set-slot-value ?record has-current-operator ?op)
                         (if (achieved ?sub ?op)
                             (DO
                              (set-slot-value ?record
                                                has-schedule-operators
                                                (remove ?op ?ops))
                              ;;Try adding same
                              (in-environment
                               ((?sub2 . (instantiate-generic-subtask
                                           ?task try-schedule-operator
                                           has-schedule-operator ?op
                                           has-schedule-focus ?focus
                                           has-schedule-value ?value
                                           has-schedule-activity-value
                                           ?activity-value
                                           has-schedule-model
                                           (the-slot-value
                                            ?state
                                            'has-schedule-model)))
                                (?result . (solve-task ?sub2)))
                                (if (achieved ?sub2 ?result)
                                    (return ?result))))
                              (return :nothing)))))))
                       :own-slots ((has-generic-subtasks '(select-schedule-operator
                                                           try-schedule-operator))))))

(def-class SELECT-SCHEDULE-OPERATOR (goal-specification-task) ?task
  ((has-input-role :value has-schedule-operators
                   :value has-schedule-focus)
   (has-output-role :value has-selected-operator)
   (has-schedule-operators :type list)
   (has-schedule-focus :type schedule-focus)
   (has-selected-operator :type schedule-operator)
   (has-goal-expression :value (kappa (?task ?op)
                                       (and (schedule-operator ?op)
                                             (has-selected-operator ?task ?op))))))

(def-class DEFAULT-OPERATOR-SELECTION (primitive-method) ?psm
  ((has-body :value (lambda (?psm)
                      (first (role-value ?psm
                                          'has-schedule-operators))))
   :own-slots ((tackles-task-type select-schedule-operator)))

```

```

(def-class TRY-SCHEDULE-OPERATOR (goal-specification-task)
  ((has-input-role :value has-schedule-operator
                   :value has-schedule-focus
                   :value has-schedule-model
                   :value has-schedule-value
                   :value has-schedule-activity-value)
   (has-output-role :value generates-schedule-state)
   (has-schedule-operator :type schedule-operator)
   (has-schedule-focus :type schedule-focus)
   (has-schedule-model :type schedule-model)
   (has-schedule-value :type schedule-value)
   (has-schedule-activity-value :type activity-value)
   (generates-schedule-state :type schedule-state)
   (has-goal-expression :value (kappa (?task ?s)
                                       (and (schedule-state ?s)
                                             (generates-schedule-state ?task ?s))))))

(def-class TRY-SCHEDULE-EXTENSION-TIME-RANGE-OPERATOR (primitive-method)
  ((has-body :value (lambda (?psm)
                      (in-environment
                       ((?sc . (role-value ?psm 'has-schedule-model))
                        (?focus . (role-value ?psm 'has-schedule-focus))
                        (?value . (role-value ?psm 'has-schedule-value))
                        (?activity-value . (role-value
                                           ?psm 'has-schedule-activity-value))
                        (?value1 . (apply-schedule-extension-time-range-operator
                                   ?focus ?sc
                                   (role-value ?psm 'has-schedule-operator))))
                       (if (not (= ?value1 :nothing))
                           (achieve-generic-subtask
                            ?psm
                            new-schedule-state
                            has-schedule-model
                            (cons
                             (cons ?focus '(?value ?activity-value ?value1))
                             ?sc)))))))
   :own-slots ((tackles-task-type try-schedule-operator)))

(def-function APPLY-SCHEDULE-EXTENSION-TIME-RANGE-OPERATOR (?j ?sc ?op)
  :constraint (and (job ?j)
                  (schedule-model ?sc)
                  (schedule-extension-time-range-operator ?op))
  :body (call (the ?body
                  (has-body ?op ?body)) ?j ?sc))

(def-relation SCHEDULE-FOCUS-ORDER (?x ?c)
  :constraint (and (schedule-focus ?x)
                  (schedule-focus ?c)
                  (not (= ?x ?c))))

(tell (defines-partial-order schedule-focus-order))

(def-function SELECT-MOST-PREFERRED-FOCUS (?l ?rel)
  :body (the ?focus
          (and (member ?focus ?l)
               (not (exists ?focus2
                           (and (member ?focus2 ?l)
                                (> ?focus2 ?focus)
                                (holds ?rel ?focus2 ?focus)))))))

(tell (use-method consistent-maximal-state-selection
                  choose-schedule-state
                  generic-psm-for-scheduling))
;;;;;;;;;;

(def-class GENERIC-SCHEDULE-APPLICATION (application)
  "This class needs to be instantiated for solving an application. This class explicitly
  states, which task needs to be solved (which in the case of this library the scheduling
  task) and which method to be used in order to solve the task.
  )

(def-class SCHEDULE-METHOD (problem-solving-method)
  ((applicable-to-task-type :value scheduling-task)
   (has-input-role :value has-schedule-operators)
   (has-schedule-operators :type schedule-operator))

;;;;;;;;;;

```

# Appendix 3

## A COMPLETE SPECIFICATION OF THE SIMPLE TIME ONTOLOGY

---

```
;;; Mode: Lisp; Package: ocml

;;; The Open University

(in-package "OCML")

(in-ontology simple-time)

(def-class YEAR-IN-TIME ()?x
  "A year-in-time must be an integer and integer can be a year-in-time"
  :iff-def (integer ?x))

(def-class MONTH-IN-TIME ()?mit
  "A month-in-time is an integer in the interval 1-12"
  :iff-def (and (integer ?x) (< ?x 12) (> ?x 0)))

(def-class DAY-IN-TIME ()?x
  "A day-in-time is an integer in the interval 1-31"
  :iff-def (and (integer ?x) (< ?x 32) (or (> ?x 0) (= ?x 1))))

(def-class HOUR-IN-TIME ()?x
  "A hour-in-time is an integer in the interval 0-23"
  :iff-def (and (integer ?x) (< ?x 24) (or (= ?x 0) (> ?x 0))))

(def-class SECOND-IN-TIME ()?x
  "A second-in-time is a integer in the interval 0-59"
  :iff-def (and (integer ?x) (< ?x 60) (or (= ?x 0) (> ?x 0))))

(def-class MINUTE-IN-TIME ()?x
  "A minute-in-time is an integer in the interval 0-59"
  :iff-def (and (integer ?x) (< ?x 60) (or (= ?x 0) (> ?x 0))))

(def-class TIME-ENTITY () ?te
)

(def-class TIME-POINT () ?tp
  ((second-of :type second-in-time :max-cardinality 1)
   (minute-of :type minute-in-time :max-cardinality 1)
   (hour-of :type hour-in-time :max-cardinality 1)
   (day-of :type day-in-time :max-cardinality 1)
   (month-of :type month-in-time :max-cardinality 1)
   (year-of :type year-in-time :max-cardinality 1))
  :constraint (and (not (and (month-of ?x 2)
                             (> (the ?day (day-of ?x ?day))
                                29)))
                  (not (and (member-of ?x (4 6 9 11))
                             (> (the ?day (day-of ?x ?day))
                                30)))))

(def-relation IDLE-TIME-POINT (?tp)
  :constraint (time-point ?tp)
  :iff-def (and (= (second-of-tp ?tp) 0)
                (= (minute-of-tp ?tp) 0)
                (= (hour-of-tp ?tp) 0)
                (= (day-of-tp ?tp) 0)
                (= (month-of-tp ?tp) 0)
                (= (year-of-tp ?tp) 0)))

(def-function SECOND-OF-TP (?tp)
  :constraint (time-point ?tp)
  :body (the ?second (second-of ?tp ?second)))

(def-function MINUTE-OF-TP (?tp)
  :constraint (time-point ?tp)
  :body (the ?minute (minute-of ?tp ?minute)))
```

```

(def-function HOUR-OF-TP (?tp)
  :constraint (time-point ?tp)
  :body (the ?hour (hour-of ?tp ?hour)))

(def-function DAY-OF-TP (?tp)
  :constraint (time-point ?tp)
  :body (the ?day (day-of ?tp ?day)))

(def-function MONTH-OF-TP (?tp)
  :constraint (time-point ?tp)
  :body (the ?month (month-of ?tp ?month)))

(def-function YEAR-OF-TP (?tp)
  :constraint (time-point ?tp)
  :body (the ?year (year-of ?tp ?year)))

(def-class INTERVAL () ?int
  "An interval is a period of time elapsed between the start of an event and end of an
  event. The start of an event is precedes the end of an event. (Ref. J.F.Allen (1983),
  Maintaining knowledge about temporal intervals)."
  ((has-start-time :type time-point :max-cardinality 1)
   (has-end-time :type time-point :max-cardinality 1)
   (has-unit-of-measure :type unit-of-measure))
  :constraint (precedes (the-slot-value ?int has-start-time)
                     (the-slot-value ?int has-end-time)))

(def-function TIME-INTERVAL-DURATION (?interval) -> ?duration
  :constraint (and (interval ?interval)
                  (duration ?duration))
  :body (time-point-difference (the ?et (has-end-time ?interval ?et))
                              (the ?st (has-start-time ?interval ?st))))

(def-class TIME-RANGE (interval) ?tr
)

(def-function TIME-RANGE-DURATION (?tr) -> ?duration
  :constraint (and (time-range ?tr)
                  (duration ?duration))
  :body (time-point-difference (the ?et (has-end-time ?tr ?et))
                              (the ?st (has-start-time ?tr ?st))))

(def-class THING ()
)

(def-class INTANGIBLE-THING (thing)
  "This comes from HPKB upper level. The collection of things that are not physical--are not
  made of, or encoded in, matter. Every collection is an intangibile (even if its instances
  are tangible), and so are some Individual. Caution: do not confuse 'tangibility' with
  'perceivability'-- humans can perceive light even though it's intangible-- at least in a
  sense.")

(def-class TANGIBLE (thing)
  "Something which is not tangible.")

(def-axiom TANGIBLE-AND-INTANGIBLE-THINGS-ARE-DISJOINT
  (exhaustive-subclass-partition (set-of tangible-thing intangible-thing)))

(def-class QUANTITY (intangible-thing) ?qun
  ((has-unit-of-measure :type unit-of-measure)
   (has-magnitude :type number)))

(def-class UNIT-OF-MEASURE (intangible-thing)
  "Any kind of unit of measure, meter, dollar, kilogram, a month, a day, a year etc..")

(def-class DURATION (quantity) ?d
)

(def-function MAGNITUDE-OF-DURATION (?dur) -> ?mag
  :constraint (and (duration ?dur)
                  (number ?mag))
  :body (the ?mag (has-magnitude ?dur ?mag)))

(def-function UNIT-OF-DURATION (?dur) -> ?uom
  :constraint (and (duration ?dur)
                  (unit-of-measure ?uom))
  :body (the ?uom (has-unit-of-measure ?dur ?uom)))

```

```

(def-class CALENDAR-DATE (time-point)
  "A calendar date is a time point in which month, day and year have been specified"
  ((day-of :type day-in-time :cardinality 1)
   (month-of :type month-in-time :cardinality 1)
   (year-of :type year-in-time :cardinality 1)))

(def-function UNIVERSAL-TIME-ENCODER (?tp)
  "This function encodes the standard structure of time-point into universal-time structure."
  :constraint (time-point ?tp)
  :lisp-fun '(lambda (?tp)
    (encode-universal-time (the-slot-value ?tp 'second-of)
                          (the-slot-value ?tp 'minute-of)
                          (the-slot-value ?tp 'hour-of)
                          (the-slot-value ?tp 'day-of)
                          (the-slot-value ?tp 'month-of)
                          (the-slot-value ?tp 'year-of))))

(def-class UNIVERSAL-TIME () ?x
  :constraint (integer ?x))

(def-function DECODE-TIME-POINT-FROM-UNIVERSAL-TIME (?ut)
  :constraint (universal-time ?ut)
  :lisp-fun '(lambda (?ut)
    (multiple-value-bind
      (second minute hour day month year ignore1 ignore2 ignore3)
      (decode-universal-time ?ut)
      (name
       (define-domain-instance (gentemp "TIME-POINT") 'time-point
        ~{(second-of ,second)
          (minute-of ,minute)
          (hour-of ,hour)
          (day-of ,day)
          (month-of ,month)
          (year-of ,year)}))))))

(def-function TIME-POINT-DIFFERENCE (?tp-1 ?tp-2)
  "This function calculates the difference of two universal-time structures."
  :constraint (and (time-point ?tp-1)
                  (time-point ?tp-2))
  :body (decode-time-point-from-universal-time
    (- (universal-time-encoder ?tp-1) (universal-time-encoder ?tp-2))))

(def-function TIME-POINT-SUM (?tp-1 ?tp-2)
  "This function calculates the sum of two universal-time structures."
  :constraint (and (time-point ?tp-1)
                  (time-point ?tp-2))
  :body (decode-time-point-from-universal-time
    (+ (universal-time-encoder ?tp-1) (universal-time-encoder ?tp-2))))

(def-relation DURATION-IS-LESS-THAN (?d1 ?d2)
  :constraint (and (duration ?d1)
                  (duration ?d2))
  :iff-def (< (the ?magnitude1 (has-magnitude ?d1 ?magnitude1))
            (the ?magnitude2 (has-magnitude ?d2 ?magnitude2))))

(def-class JOB-TIME-RANGE () ?jtr
  "It represents the time range of each job in terms of its earliest and latest start and end time."
  ((has-earliest-start-time :type time-point :min-cardinality 1)
   (has-latest-start-time :type time-point :min-cardinality 1)
   (has-earliest-end-time :type time-point :min-cardinality 1)
   (has-latest-end-time :type time-point :min-cardinality 1)
   (has-unit-of-measure :type unit-of-measure))
  :iff-def (or (precedes (the ?est (has-earliest-start-time ?jtr ?est))
                        (the ?eet (has-earliest-end-time ?jtr ?eet)))
              (precedes (the ?lst (has-latest-start-time ?jtr ?lst))
                        (the ?let (has-latest-end-time ?jtr ?let)))))

(def-function JOB-TIME-RANGE-DURATION (?jtr) -> ?job-duration
  :constraint (and (job-time-range ?jtr)
                  (duration ?job-duration))
  :body (- (the-slot-value ?jtr has-latest-end-time)
           (the-slot-value ?jtr has-earliest-start-time)))

(def-function JOB-DURATION-QUANTITY (?job-duration) -> ?magnitude
  :constraint (and (duration ?job-duration)
                  (number ?magnitude))
  :body (the ?magnitude (has-magnitude ?job-duration ?magnitude)))

```

```

(def-instance second unit-of-measure)

(def-instance minute unit-of-measure)

(def-instance hour unit-of-measure)

(def-instance day unit-of-measure)

(def-instance month unit-of-measure)

(def-instance year unit-of-measure)

;;;;;;;;;;;;;
;;;**;;Following are the useful relations for the Time-Ranges;;;*;;
;;;;;;;;;;;;;

(def-relation PRECEDES (?time-point-1 ?time-point-2)
  "This relation states that a ?time-point-1 precedes a time-point ?time-point-2."
  :constraint (and (time-point ?time-point-1)
                   (time-point ?time-point-2))
  :iff-def (< (universal-time-encoder ?time-point-1)
              (universal-time-encoder ?time-point-2)))

(def-relation FOLLOWS (?time-point-1 ?time-point-2)
  "This relation states that a time-point ?time-point-2 follows a time-point ?time-
point-1."
  :constraint (and (time-point ?time-point-1)
                   (time-point ?time-point-2))
  :iff-def (precedes ?time-point-2 ?time-point-1))

(def-relation TIME-POINTS-EQUAL (?time-point-1 ?time-point-2)
  :constraint (and (time-point ?time-point-1)
                   (time-point ?time-point-2))
  :iff-def (and (= (minute-of ?time-point-1)
                   (minute-of ?time-point-2))
                (= (second-of ?time-point-1)
                   (second-of ?time-point-2))
                (= (hour-of ?time-point-1)
                   (hour-of ?time-point-2))
                (= (day-of ?time-point-1)
                   (day-of ?time-point-2))
                (= (month-of ?time-point-1)
                   (month-of ?time-point-2))
                (= (year-of ?time-point-1)
                   (year-of ?time-point-2))))

;;;These are BASIC relations;;;

(def-relation BEFORE (?time-range-1 ?time-range-2)
  "It means time-range-1 is before the time-range-2."
  :constraint (and (time-range ?time-range-1)
                   (time-range ?time-range-2))
  :iff-def (precedes (the ?et (has-end-time ?time-range-1 ?et))
                    (the ?st (has-start-time ?time-range-2 ?st))))

(def-relation AFTER (?time-range-1 ?time-range-2)
  "It means time-range-1 is after the time-range-2."
  :constraint (and (time-range ?time-range-1)
                   (time-range ?time-range-2))
  :iff-def (precedes (the ?et (has-end-time ?time-range-2 ?et))
                    (the ?st (has-start-time ?time-range-1 ?st))))

```

```

(def-relation IS-AFTER (?time-range-2 ?time-range-1)
  "It means that time-range-2 starts after the time-range-1 is finished."
  :constraint (and (time-range ?time-range-1)
                   (time-range ?time-range-2))
  :iff-def (follows (the ?st (has-start-time ?time-range-2 ?st))
             (the ?et (has-end-time ?time-range-1 ?et))))

(def-relation MEETS (?time-range-1 ?time-range-2)
  "It means that time-range-2 starts at the same time when time-range-1 ends."
  :constraint (and (time-range ?time-range-1)
                   (time-range ?time-range-2))
  :iff-def (time-points-equal (the ?et (has-end-time ?time-range-1 ?et))
                              (the ?st (has-start-time ?time-range-2 ?st))))

(def-relation OVERLAPS (?time-range-1 ?time-range-2)
  "It means that two time-ranges overlaps with each other."
  :constraint (and (time-range ?time-range-1)
                   (time-range ?time-range-2))
  :iff-def (and (precedes (the ?st-1 (has-start-time ?time-range-1 ?st-1))
                        (the ?st-2 (has-start-time ?time-range-2 ?st-2)))
              (follows (the ?et-1 (has-end-time ?time-range-1 ?et-1))
                       (the ?st-2 (has-start-time ?time-range-2 ?st-2)))
              (precedes (the ?et-1 (has-end-time ?time-range-1 ?et-1))
                        (the ?et-2 (has-end-time ?time-range-2 ?et-2))))

(def-relation STARTS-SIMULTANEOUSLY (?time-range-1 ?time-range-2)
  "It means that both the time-ranges starts at the same time."
  :constraint (and (time-range ?time-range-1)
                   (time-range ?time-range-2))
  :iff-def (time-points-equal (the ?st-1 (has-start-time ?time-range-1 ?st-1))
                              (the ?st-2 (has-start-time ?time-range-2 ?st-2))))

(def-relation FINISHES-SIMULTANEOUSLY (?time-range-1 ?time-range-2)
  "It means that both the time-ranges finishes at the same time but time-range-1 starts after
time-range-2."
  :constraint (and (time-range ?time-range-1)
                   (time-range ?time-range-2))
  :iff-def (time-points-equal (the ?et-1 (has-end-time ?time-range-1 ?et-1))
                              (the ?et-2 (has-end-time ?time-range-2 ?et-2))))

(def-relation TIME-RANGE-EQUALS (?time-range-1 ?time-range-2)
  "It means that both the time-ranges starts and finishes at the same time."
  :constraint (and (time-range ?time-range-1)
                   (time-range ?time-range-2))
  :iff-def (and (time-point-equals (the ?st-1 (has-start-time ?time-range-1 ?st-1))
                                   (the ?st-2 (has-start-time ?time-range-2 ?st-2)))
              (time-point-equals (the ?et-1 (has-end-time ?time-range-1 ?et-1))
                                   (the ?et-2 (has-end-time ?time-range-2 ?et-2))))

(def-relation TIME-POINT-WITHIN-INTERVAL (?tp ?interval)
  :constraint (and (time-point ?tp)
                   (interval ?interval))
  :iff-def (and (or (follows (the-slot-value ?interval has-end-time)
                             (the-slot-value ?tp second-of))
                    (follows (the-slot-value ?interval has-end-time)
                             (the-slot-value ?tp minute-of))
                    (follows (the-slot-value ?interval has-end-time)
                             (the-slot-value ?tp hour-of))
                    (follows (the-slot-value ?interval has-end-time)
                             (the-slot-value ?tp day-of))
                    (follows (the-slot-value ?interval has-end-time)
                             (the-slot-value ?tp month-of))
                    (follows (the-slot-value ?interval has-end-time)
                             (the-slot-value ?tp year-of)))
              (or (precedes (the-slot-value ?interval has-start-time)
                             (the-slot-value ?tp second-of))
                  (precedes (the-slot-value ?interval has-start-time)
                             (the-slot-value ?tp minute-of))
                  (precedes (the-slot-value ?interval has-start-time)
                             (the-slot-value ?tp hour-of))
                  (precedes (the-slot-value ?interval has-start-time)
                             (the-slot-value ?tp day-of))
                  (precedes (the-slot-value ?interval has-start-time)
                             (the-slot-value ?tp month-of))
                  (precedes (the-slot-value ?interval has-start-time)
                             (the-slot-value ?tp year-of))))))

```

```

;;;;;;;;;;
;;;These are derived relations;;;
;;;;;;;;;;

(def-relation IS-DURING (?time-range-1 ?time-range-2)
  "It means that time-range-2 is in between (during) the the start and end time of time-range-1."
  :constraint (and (time-range ?time-range-1)
                    (time-range ?time-range-2))
  :iff-def (and (precedes (the ?st-1 (has-start-time ?time-range-1 ?st-1))
                    (the ?st-2 (has-start-time ?time-range-2 ?st-2)))
                (follows (the ?et-1 (has-end-time ?time-range-1 ?et-1))
                          (the ?et-2 (has-end-time ?time-range-2 ?et-2)))))

(def-relation JOB-ACTIVITY-TIME-RANGE-IS-DURING (?jtr-1 ?jtr-2)
  :constraint (and (job-time-range ?jtr-1)
                    (time-range ?jtr-2))
  :iff-def (and (precedes (the ?est-1 (has-earliest-start-time ?jtr-1 ?est-1))
                    (the ?est-2 (has-earliest-start-time ?jtr-2 ?est-2)))
                (follows (the ?let-1 (has-latest-end-time ?jtr-1 ?let-1))
                          (the ?let-2 (has-latest-end-time ?jtr-2 ?let-2)))))

(def-relation BEFORE-OR-EQUAL (?time-range-1 ?time-range-2)
  "It says that either one time range is before the other or is equal to the other time range."
  :constraint (and (time-range ?time-range-1)
                    (time-range ?time-range-2))
  :iff-def (or (before ?time-range-1 ?time-range-2)
                (meets ?time-range-1 ?time-range-2)))

(def-relation AFTER-OR-EQUAL (?time-range-1 ?time-range-2)
  "It says that either one time range is after the other or is equal to the other time range."
  :constraint (and (time-range ?time-range-1)
                    (time-range ?time-range-2))
  :iff-def (or (after ?time-range-1 ?time-range-2)
                (meets ?time-range-1 ?time-range-2)))

(def-relation IS-AFTER-THAN (?time-range-1 ?time-range-2)
  "It is true when one time range is after the otehr time range."
  :constraint (and (time-range ?time-range-1)
                    (time-range ?time-range-2))
  :iff-def (is-after ?time-range-2 ?time-range-1))

(def-relation DURING-OR-EQUAL (?time-range-1 ?time-range-2)
  "It is true when one time range is-during the other time range or both these time ranges starts or finishes simultaneously or they are equal to each other."
  :constraint (and (time-range ?time-range-1)
                    (time-range ?time-range-2))
  :iff-def (or (is-during ?time-range-1 ?time-range-2)
                (starts-simultaneously ?time-range-1 ?time-range-2)
                (finishes-simultaneously ?time-range-1 ?time-range-2)
                (time-range-equals ?time-range-1 ?time-range-2)))

(def-relation JOB-TIME-RANGE-DURING-OR-EQUAL (?jtr ?time-range)
  :constraint (and (job-time-range ?jtr)
                    (time-range ?time-range))
  :iff-def (or (and (< (has-earliest-start-time ?jtr ?est)
                       (has-start-time ?time-range ?st))
                    (< (has-latest-end-time ?jtr ?let)
                       (has-end-time ?time-range ?et)))
                (and (= (has-earliest-start-time ?jtr ?est)
                       (has-start-time ?time-range ?st))
                    (= (has-latest-end-time ?jtr ?let)
                       (has-end-time ?time-range ?et)))))

(def-relation EQUAL-LENGTH-TIME-RANGES (?time-range-1 ?time-range-2)
  "The two time ranges are of equal length if they both start at the ssame time and finsh at the same time as well."
  :constraint (and (job-time-range ?time-range-1)
                    (job-time-range ?time-range-2))
  :iff-def (and (time-points-equal (the-slot-value ?time-range-1 has-earliest-start-time)
                                    (the-slot-value ?time-range-2 has-earliest-start-time))
                (time-points-equal (the-slot-value ?time-range-1 has-latest-start-time)
                                    (the-slot-value ?time-range-2 has-latest-start-time))
                (time-points-equal (the-slot-value ?time-range-1 has-earliest-end-time)
                                    (the-slot-value ?time-range-2 has-earliest-end-time))
                (time-points-equal (the-slot-value ?time-range-1 has-latest-end-time)
                                    (the-slot-value ?time-range-2 has-latest-end-time))))

```

```

(def-relation OVERLAPS-OR-MEETS (?time-range-1 ?time-range-2)
  "It is true when two time ranges either overlaps with each other or meets each other."
  :constraint (and (time-range ?time-range-1)
                   (time-range ?time-range-2))
  :iff-def (or (overlaps ?time-range-1 ?time-range-2)
               (meets ?time-range-1 ?time-range-2)))

(def-relation OVERLAPS-OR-EQUALS (?time-range-1 ?time-range-2)
  "It is true when two time ranges either overlaps with each other and are equal to each other."
  :constraint (and (time-range ?time-range-1)
                   (time-range ?time-range-2))
  :iff-def (or (overlaps ?time-range-1 ?time-range-2)
               (time-range-equals ?time-range-1 ?time-range-2)))

(def-relation STARTS-OR-EQUAL (?time-range-1 ?time-range-2)
  "It is true when two time ranges either starts simulataneously or are equal to each other."
  :constraint (and (time-range ?time-range-1)
                   (time-range ?time-range-2))
  :iff-def (or (starts-simultaneously ?time-range-1 ?time-range-2)
               (time-range-equals ?time-range-1 ?time-range-2)))

(def-relation FINISHES-OR-EQUALS (?time-range-1 ?time-range-2)
  "It is true when two time ranges finishes simultaneously or are equal to each other."
  :constraint (and (time-range ?time-range-1)
                   (time-range ?time-range-2))
  :iff-def (or (finishes-simultaneously ?time-range-1 ?time-range-2)
               (time-range-equals ?time-range-1 ?time-range-2)))

(def-relation DISJOINT-TIME-RANGES (?time-range-1 ?time-range-2)
  "It is true if either time-range-1 is before time-range-2 or time-range-2 is before time-range-1."
  :constraint (and (time-range ?time-range-1)
                   (time-range ?time-range-2))
  :iff-def (or (before ?time-range-1 ?time-range-2)
               (before ?time-range-2 ?time-range-1)))

(def-relation TIME-RANGES-NOT-EXCEED (?job-time-range ?time-range)
  :constraint (and (exists ?j (job ?j has-time-range ?job-time-range))
                   (time-range ?tr))
  :iff-def (and (precedes (the ?est (has-earliest-start-time ?job-time-range ?est))
                           (the ?et (has-end-time ?time-range ?et)))
                (follows (the ?let (has-latest-end-time ?job-time-range ?let))
                           (the ?st (has-start-time ?tr ?st)))
                (precedes (the ?let (has-latest-end-time ?job-time-range ?let))
                           (the ?et (has-end-time ?time-range ?et)))))

(def-relation TIME-RANGES-INTERSECT (?jtr ?tr)
  :constraint (and (job-time-range ?jtr)
                   (time-range ?tr))
  :iff-def (and (follows (the ?est (has-earliest-start-time ?jtr ?est))
                           (the ?st (has-start-time ?tr ?st)))
                (follows (the ?lst (has-latest-start-time ?jtr ?lst))
                           (the ?st (has-start-time ?tr ?st)))
                (precedes (the ?eet (has-earliest-end-time ?jtr ?eet))
                           (the ?et (has-end-time ?tr ?et)))
                (precedes (the ?let (has-latest-end-time ?jtr ?let))
                           (the ?et (has-end-time ?tr ?et)))))

(def-relation DUE-DATE-EARLIER-THAN-OTHER (?dd1 ?dd2)
  "It says that if each of the slot value of due-date-1 precedes every slot-value of due-date-2 then due-date-1 is earlier-than due-date-2."
  :constraint (and (calendar-date ?dd1)
                   (calendar-date ?dd2))
  :iff-def (and (precedes (the-slot-value ?dd1 day-of)
                           (the-slot-value ?dd2 day-of))
                (precedes (the-slot-value ?dd1 month-of)
                           (the-slot-value ?dd2 month-of))
                (precedes (the-slot-value ?dd1 year-of)
                           (the-slot-value ?dd2 year-of))))

(def-relation DUE-DATE-LATER-THAN-OTHER (?dd2 ?dd1)
  "It says that is each of the slot value of due-date-2 follows every slot-value of due-date-1 then due-date-2 is later than due-date-1."
  :constraint (and (calendar-date ?dd1)
                   (calendar-date ?dd2))
  :iff-def (and (follows (the-slot-value ?dd2 day-of)
                           (the-slot-value ?dd1 day-of))
                (follows (the-slot-value ?dd2 month-of)
                           (the-slot-value ?dd1 month-of))
                (follows (the-slot-value ?dd2 year-of)
                           (the-slot-value ?dd1 year-of))))

```

```

////////////////////////////////////

```

;;The following relations are defined for the time intervals exactly as it is described in J F Allen's paper.

```
(def-relation TIME-INTERVAL-BEFORE (?ti1 ?ti2)
:constraint (and (interval ?ti1)
                  (interval ?ti2))
:iff-def (and (precedes (the-slot-value ?ti1 has-start-time)
                        (the-slot-value ?ti2 has-start-time))
              (precedes (the-slot-value ?ti1 has-end-time)
                        (the-slot-value ?ti2 has-end-time))))

(def-relation TIME-INTERVAL-EQUAL (?ti1 ?ti2)
:constraint (and (interval ?ti1)
                  (interval ?ti2))
:iff-def (and (time-points-equal (the-slot-value ?ti1 has-start-time)
                                  (the-slot-value ?ti2 has-start-time))
              (time-points-equal (the-slot-value ?ti1 has-end-time)
                                  (the-slot-value ?ti2 has-end-time))))

(def-relation TIME-INTERVAL-MEETS (?ti1 ?ti2)
:constraint (and (interval ?ti1)
                  (interval ?ti2))
:iff-def (time-points-equal (the-slot-value ?ti2 has-end-time)
                            (the-slot-value ?ti1 has-start-time)))

(def-relation TIME-INTERVAL-OVERLAPS (?ti1 ?ti2)
:constraint (and (interval ?ti1)
                  (interval ?ti2))
:iff-def (and (precedes (the-slot-value ?ti1 has-start-time)
                        (the-slot-value ?ti2 has-start-time))
              (follows (the-slot-value ?ti1 has-end-time)
                       (the-slot-value ?ti2 has-start-time))
              (precedes (the-slot-value ?ti1 has-end-time)
                        (the-slot-value ?ti2 has-end-time))))

(def-relation TIME-INTERVAL-DURING (?ti1 ?ti2)
:constraint (and (interval ?ti1)
                  (interval ?ti2))
:iff-def (and (precedes (the-slot-value ?ti1 has-start-time)
                        (the-slot-value ?ti2 has-start-time))
              (follows (the-slot-value ?ti1 has-end-time)
                       (the-slot-value ?ti2 has-end-time))))

(def-relation TIME-INTERVAL-STARTS (?ti1 ?ti2)
:constraint (and (interval ?ti1)
                  (interval ?ti2))
:iff-def (time-points-equal (the-slot-value ?ti1 has-start-time)
                            (the-slot-value ?ti2 has-start-time)))

(def-relation TIME-INTERVAL-FINISHES (?ti1 ?ti2)
:constraint (and (interval ?ti1)
                  (interval ?ti2))
:iff-def (time-points-equal (the-slot-value ?ti1 has-end-time)
                            (the-slot-value ?ti2 has-end-time)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;(Alex) Chao-Chiang Meng and Michael Sullivan (1991). Logos A Constraint-Directed;;;
;;;Reasoning Shell for Operations Management, IEEE Expert, 6(1), pp.01-16.;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(def-relation TIME-INTERVAL-ELAPSED-BY (?ti1 ?ti2)
"This relation states, if one interval precedes another interval, then, it says, by how
much time another interval succeeds the prior interval."
:constraint (and (interval ?ti1)
                  (interval ?ti2)
                  (has-start-time ?ti1 ?st1)
                  (has-end-time ?ti1 ?et1)
                  (has-start-time ?ti2 ?st2)
                  (has-end-time ?ti2 ?et2))
:iff-def (or (precedes ?et1 ?st2)
              (= ?et1
                 (+ (?st2 (= ?diff-tp
                             (time-point-difference ?et1 ?st2)))))))
```

```

(def-relation TIME-INTERVAL-DURING-DELAY-AND-LAG (?ti1 ?ti2)
  "This relation states that if two intervals are during each other, and if one interval
  delays or lag from the other interval, it states by how much margin the interval has
  delayed or lagged."
  :constraint (and (interval ?ti1)
    (has-start-time ?ti1 ?st1)
    (has-end-time ?ti1 ?et1)
    (interval ?ti2)
    (has-start-time ?ti2 ?st2)
    (has-end-time ?ti2 ?et2))
  :iff-def (and (or (follows ?st2 ?st1)
    (= ?st2
      (+ (?st1
        (= ?diff-tp
          (time-point-difference ?st2 ?st1))))))
    (or (precedes ?st2 ?et1)
      (= ?et1
        (+ (?et2
          (= ?diff-tp
            (time-point-difference ?et1 ?et2))))))))))

(def-relation TIME-INTERVAL-OVERLAP-OR-LAG (?ti1 ?ti2)
  "This relation states that if two intervals are overlapping each other, and if one interval
  lags another interval, then it says by how much margin these intervals are lagged."
  :constraint (and (interval ?ti1)
    (has-start-time ?ti1 ?st1)
    (has-end-time ?ti1 ?et1)
    (interval ?ti2)
    (has-start-time ?ti2 ?st2)
    (has-end-time ?ti2 ?et2))
  :iff-def (and (precedes ?st1 ?st2)
    (or (and (precedes ?st2 ?et1)
      (precedes ?et1 ?et2))
      (= ?et1
        (+ (?st1
          (= ?diff-tp
            (time-point-difference ?et1 ?st2))))))))))

(def-relation TIME-INTERVAL-STARTS-BY (?ti1 ?ti2)
  "This relation states that if two intervals starts at the same time, and if one interval
  finishes after another interval then it says by how much margin the interval finishes over
  other."
  :constraint (and (interval ?ti1)
    (has-start-time ?ti1 ?st1)
    (has-end-time ?ti1 ?et1)
    (interval ?ti2)
    (has-start-time ?ti2 ?st2)
    (has-end-time ?ti2 ?et2))
  :iff-def (and (time-points-equal ?st1 ?st2)
    (or (precedes ?et1 ?et2)
      (= ?et2
        (+ (?et1
          (= ?diff-tp
            (time-point-difference ?et2 ?et1))))))))))

(def-relation TIME-INTERVAL-FINISHES-DELAY (?ti1 ?ti2)
  "This relation states that if two interval finishes at the same time, but they have
  different start-time, then it says, by how much time these two intervals differ in terms of
  thri start times."
  :constraint (and (interval ?ti1)
    (has-start-time ?ti1 ?st1)
    (has-end-time ?ti1 ?et1)
    (interval ?ti2)
    (has-start-time ?ti2 ?st2)
    (has-end-time ?ti2 ?et2))
  :iff-def (and (or (follows ?st1 ?st2)
    (= ?st1
      (+ (?st2
        (= ?diff-tp
          (time-point-difference ??st1 ?st2))))))
    (time-points-equal ?et1 ?et2)))
  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

# Appendix 4

## A REFERENCE GUIDE TO OCML

Because our library is implemented by using the Operational Conceptual Modelling Language (OCML) (Motta, 1999), here we provide a reference guide to some of the important modelling constructs and the basic modelling mechanism supported by OCML.

### 4.1 Different types of constructs in OCML

In OCML the following three types of constructs are supported: functional terms, control terms, and logical expressions. These are discussed in the following bullet points.

- **Functional terms:** It is used to specify an object in the current domain of interest. The functional term can be a constant, a variable (it is represented as a Lisp symbol with the question mark prefix, e.g. ?x), a string (it is represented by a double quote "a generic library of scheduling"), a function application (it is represented by means of the Lisp macro def-function), or it can be constructed by special term constructor. The special term constructs can be one of the following: if, cond, the, setofall, findall, quote, and in-environment;
- **Control terms:** Modelling the problem solving behaviour involves more than making statements and describing entities in the world. Control regimes are required to specify actions and describe the order in which these are executed. OCML supports the specification of sequential, iterative, and conditional control structures by means of a number of control term constructors such as repeat, loop, do, if and cond, etc;
- **Logical expressions:** OCML also supports a mechanism for specifying the logical expressions. Some of the typical logical operators that can be used to construct the logical expressions in OCML are the following ones: and, or, not, =>, <=>, and quantifiers such as forall and exists. The simplest type of logical expression in OCML is a relation expression which is specified by means of Lisp macro def-relation.

### 4.2 Basic modelling in OCML

OCML provide mechanisms for describing various types of primitives for modelling which are as follows: classes, instances, relations, functions, rules, procedures, and axioms.

### 4.2.1 OCML classes

OCML supports the specification of classes and instances and the inheritance of slots and values in terms of *isa* hierarchy. The classes in OCML are represented in terms of the Lisp macro called, `def-class`. It takes as an argumentation a name of a class, a list of superclasses if the class is inheriting the specification from other classes, and a list of slot specifications. In the case where the class is a goal-specification task (cf. Appendix 1) then the input and output role specifies the input knowledge roles required by the class and output role specifies what the class is expected to produce as an output to the class. The following box shows the examples of the class specification in OCML.

```
(def-class JOB () ?j
  ((has-activity :type list :cardinality 1
                 :documentation "It states that each job has a list
                               of activities associated with it.")
   (requires-resource :type resource :min-cardinality 1)
   (requires-resource-type :type resource-type :min-cardinality 1)
   (has-time-range :type job-time-range :max-cardinality 1)
   (has-due-date :type calendar-date :max-cardinality 1)
   (has-duration :type duration :max-cardinality 1)
   (has-load :type integer :default-value 1)))
:iff-def (exists ?task (and (scheduling-task ?task)
                             (member ?j (role-value ?task has-jobs)))))

(def-class nimbus-1-job (job))
```

OCML provides a support for the usual slot specification that is found in frame-based representation.

- **:value:** A value that is inherited by all instances of class;
- **:default-value:** A value that is inherited by all instances of a class unless overridden by other values;
- **:type:** The value of this option should be another class, C and all values of the associated slot should be instances of C.
- **:max-cardinality:** The maximum numbers of slot values allowed for a slot.
- **:min-cardinality:** The minimum numbers of slot values required for a slot.
- **:cardinality:** The number of slot values required for a slot. This option subsumes both maximum and minimum cardinality values.
- **:documentation:** It represents a documentation describing a slot.

### 4.2.2 OCML instances

OCML instances are the members of a class and they are specified in terms of a Lisp macro `def-instance`. It takes as arguments the name of the instance, the parent of the instance (i.e., the most specific class to which the instance belongs to), optional documentation, and a number of slot-value pairs. As it has been shown in the following box the slot of an instance can have multiple values, e.g. `has-activities`. The follows box shows the example of OCML instance.

```
(def-instance nimbus-1 nimbus-1-job
  ((has-activities '(nimbus-1-communication-1 nimbus-1-communication-2
                    nimbus-1-communication-3 nimbus-1-communication-4))
   (requires-resource '(low-range-antenna))
   (has-time-range nimbus-1-time-range)
   (has-duration 60-minute-duration)))
```

### 4.2.3 OCML relations

In OCML relations allow the users to define labelled n-ary relationships between different entities and the relations are specified in terms of a Lisp macro called `def-relation`. It takes as an argument the name of a relation, its argument schema, optional documentation, and a number of relation options. The relation options in particular not only specify the formal semantics of a relation but it also provides operational nature of OCML. The following bullet points discuss the relation options in OCML.

- **:iff-def** - It specifies both sufficient and necessary conditions for the relation to hold for a given set of arguments. It provides a support for both constraint checking as well as proof mechanism;
- **:sufficient** - It specifies a sufficient condition for the relation to hold for a given set of arguments. It also provides a support for the proof mechanism but does not support constraint checking;
- **:constraint** - It specifies an expression which follows from the definition of the relation and must be true for each instance of a relation. It provides a support for constraint checking but does not provide a support for proof mechanism;
- **:def** - This is for the compatibility with Ontolingua. It specifies a constraint which is also meant to provide a partial definition of a relation. It provides a support for constraint checking but does not provide a support for proof mechanism;
- **:axiom-def** - A statement which mentions the relation to which it is associated. It provides a mechanism to associate theory axioms with specific relation;
- **:prove-by** - It is used in order to provide a support for the proof mechanism but does not support the constraint checking;
- **:lisp-fun** - It is used in order to provide a support for the proof mechanism but does not support the constraint checking.

The following box shows the OCML definition to specify the relations.

```
(def-relation ACTIVITY-PRECEDES (?a1 ?a2)
  :constraint (and (activity ?a1)
                   (activity ?a2))
  :iff-def (and (has-time-range ?a1 ?jtr-a1)
                (has-time-range ?a2 ?jtr-a2)
                (has-duration ?a1 ?d1)
                (<= (time-point-sum (the-slot-value ?jtr-a1 has-earliest-start-
                                                time)
                                     (magnitude-of-duration ?d1))
                    (the-slot-value ?jtr-a2 has-earliest-start-time)))
  :axiom-def (defines-partial-order activity-precedes))
```

In some cases we may want to use the a keyword only for specification and not operationally and to deal with such kind of situations OCML provides a meta-keyword called `:no-op`, which specifies that the enclosed relation only plays a specification role. The following box shows the `:no-op` specification in OCML.

```
(def-class SCHEDULE-EXTENSION-ACTIVITY-OPERATOR-BODY (lambda-expression) ?x
  :no-op (:constraint (and (nth-domain ?x 1 job)
                          (nth-domain ?x 2 ?sc)
                          (=> (= ?z (call ?x ?j))
                              (and (has-activities ?j ?list)
                                   (member ?z ?list))))))
```

#### 4.2.4 OCML functions

The functions in general and in OCML in particular define a mapping between a list of input arguments and its output argument. The functions are applied to ground terms to generate function values. In OCML functions are specified by the Lisp macro called `def-function`, which takes as an argument the name of a function, its argument list, an optional variable indicating the output (it is represented as follows: `-> ?c`), optional documentation, and function specification options such as `:def`, `:constraint`, `:body`, and `:lisp-fun`. The most interesting function specification options are the `:body` and `:lisp-fun`. The former specifies a functional term which is evaluated in an environment in which the variables in the function are bound to the actual arguments, while the latter makes it possible to evaluate an OCML function by means of a procedural attachment. The following box shows an example of the OCML function.

```
(def-function ALL-ASSIGNABLE-JOBS (?js ?sc) -> ?x
  :constraint (and (list ?js)
                  (every ?js job)
                  (schedule-model ?sc))
  :body (setofall ?x (and (member ?x ?js)
                        (unassigned-job ?x ?sc)
                        (job-assignable ?x ?sc))))

(def-function JOB-TIME-RANGE-DURATION (?j ?jtr) -> ?time-point
  :constraint (and (job ?j)
                  (has-time-range ?j ?jtr)
                  (job-time-range ?jtr))
  :body (- (the-slot-value ?jtr has-latest-end-time)
          (the-slot-value ?jtr has-earliest-start-time)))
```

#### 4.2.5 OCML rules: rule-based reasoning

OCML also supports the specification of backward and forward rules. The former consist of number of backward clauses and each of these backward clauses specifies the different

goal-subgoal decomposition. While carrying out a proof by means of a backward clause OCML interpreter try to prove the relevant goal by firing the clauses in the order in which they are listed in the rule definition. The following box shows the example of OCML backwards rule.

```
(def-rule possible-resources-1
  ((possible-resources-for-job ?x ?r)
   if
    (nimbus-1-job ?x)
    (low-range-antenna-resource ?r)
    (handles-job ?r ?x)
    (requires-resource ?x ?r)))
```

The forwards rule consists of zero or more antecedents and one or more consequents. Antecedents are restricted to relation expressions, while any logical expression can be a consequent. In OCML when the forward rule is executed then each consequent is treated as a goal to be proven and tries to prove them until one fails. The following box shows a representation of the forward rule in OCML.

```
(def-rule Nimbus-1-requires-low-range-antenna
  (requires-resource ?n1 ?ls)
  then
  (exec (tell (has-job-belonging ?ls ?n1)))
  (exec (output "requires a resource ~S ~S" ?n1 ?ls)))
```

#### 4.2.6 OCML procedures

In OCML, procedures define actions or sequences of actions which cannot be characterised as functions between input and output arguments. For instance, the example shown in the following box is the Base Ontology definition specified to set the value of a slot. This includes a `unassert` statement, which first removes any existing values from the slot, and uses a `tell` statement to add a new value. The `tell` and `unassert` are OCML procedures themselves, where the former takes a ground logical expression and adds it to the current model and the latter takes a relation expression and removes from the current model all assertions which match it.

```
(def-procedure set-slot-value (?i ?s ?v)
  :constraint (and (instance-of ?i ?c)
                  (slot-of ?s ?c))
  :body (do (unassert (list-of ?s ?i ?any))
            (tell (list-of ?s ?i ?v))))
```

### 4.3 Summary

The primary role of OCML language is to provide operational knowledge modelling facilities. In a nutshell, it provides support for functional and control statements as well as the proof system which integrates inheritance with backward chaining, function evaluation, and procedural attachments. Because the main objective of OCML is to give operational support, it aims to support different styles of knowledge modelling such as informal, formal, and operational. Finally, OCML provides support for export mechanisms to other

representations, including Ontolingua (Farquhar *et al.*, 1997) and OWL (McGuinness and Harmelen, 2004).