Universidade Nova de Lisboa

Faculdade de Ciências e Tecnologia

Departamento de Informática

Dissertação de Mestrado

Model Driven Development Implementation of a Control Systems User Interfaces Specification Tool

Vasco Nuno da Silva de Sousa

Orientador: Prof. Doutor Vasco Amaral

Disertação apresentada na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa para obtenção do grau de Mestre em Engenharia Informática.

Lisboa

20 de Fevereiro de 2009

Resumo

Para resolver o crescente problema de incremento de compexidade dos sistemas de software (devido à complexidade do dominio do problema, complexidade do domínio da solução e a propria complexidade dos requisitos não funcionais), a comunidade de engenharia de software está a voltar-se para abordagens de Engenharia Orientada ao Modelo, onde as Linguagens de Domínio Especifico(DSL) e Ferramentas de Transformação, são aspectos essênciais.

É neste contexto que surge o projecto BATIC3S, com o fim de a partir de uma DSL de especificação, e por uso de níveis intermédios do seu mapeamento até à plataforma alvo, prototipar automaticamente Interfaces Gráficas para Sistemas de Controlo Complexos.

Havendo um desenho de DSL chamada (H)ALL já proposto para este fim, esta tese irá demonstrar uma implementação que garanta ao nível sintáctico, modelos bem formados, e ao nível semântico, que o processo de transformação não introduz falhas no modelo de destino.

Para chegar a este fim, vão-se usar ferramentas de meta-modelação de linguagens e delinear-se regras de transformação de modelos, obedecendo a uma metodologia de transformação por níveis que demonstramos ser correcto.

Palavras-chave: Transformação de Modelos, Meta-modelação, Linguagens de Domínio Especifico, Engenharia Orientada ao Modelo, Interfaces Graficas para Sistemas de Controlo Complexos

Abstract

To solve the growing problem of increasing complexity of software systems(due to the problem's domain complexity, the solution's domain complexity and the non functional requirements complexity), the software engineering community is turning to Model Driven Development approaches, where Domain Specific Languages(DSL) and Model Transformation tools are essential aspects.

It is in this context that the BATIC3S project is running, with the goal of automatically create Complex Control Systems Graphical User Interfaces, from specifications made with a DSL built for this purpose, and with the use of intermediate levels of mapping to deliver it to the target platform.

Existing a DSL design, named (H)ALL, already proposed for this purpose, this thesis will demonstrate an implementation, that guaranties at the syntactic level, the well-formedness of the created models, and at the semantic level, that the transformation process does not introduce faults into the target model.

To achieve this, a language Meta-modeling tool will be used, and a set of Model Transformation Rules will be designed, according to a layered transformation methodology, which correctness we also demonstrate.

Keywords: Model Transformation, Meta-modeling, Domain Specific Languages, Complex Control Systems Graphic User Interfaces, Model Driven Engineering

Contents

1 Introduction **1**

1.1 Context 1

1.2 Motivations 3

1.3 Proposed Solution 4

1.4 Document Structure 5

2 State of the Art **7**

2.1 Model Driven Development 7

2.1.1 Meta-modeling 8

2.1.2 Domain Specific Languages 9

2.1.3 Model Transformation 10

2.1.3.1 Graph Transformation 11

2.2 Control Systems 12

2.3 BATIC3S 12

2.3.1 (Human) Assisted Logic Language 14

2.3.2 CO-OPN 14

3 Tools **17**

3.1 Meta-Modeling Tools 17

3.1.1 Meta-Modeling Tools Decision Criteria 18

3.1.2 Meta-Modeling Tool Analysis 18

3.2 Transformation Tools and Processes 20

3.2.1 Transformation Tools Decision Criteria 22

3.2.2 Transformation Process Selection 22

3.3 Summary 23

4 Correction of the Approach **25**

4.1 Model Syntax Evaluation 25

4.1.1 Meta-Modeling Tools 25

4.1.2 Transformation Tools and Processes 26

4.2 Formalizing the Transformation 26

4.2.1 Formalisation attempts over model transformation rules 27

4.2.2 Partial order isolation of the transformation rules 27

4.2.3 Auxiliary Diagrammatic Formalism 28

4.2.4 Base Formalization Concepts 31

4.2.5 Rule Formalization 35

4.2.5.1 Left Hand Side: 35

4.2.5.2 Right Hand Side: 36

4.2.6 Rule Composition 36

4.2.7 Algorithmic Application of the Formalized Rules 38

4.2.8 Incremental Layer Specification 39

4.2.9 Semantic preservation 41

4.2.10 Confluence 42

4.2.11 Completeness 43

5 (H)ALL Language Engineering **45**

5.1 Syntax 45

5.1.1 Meta-Modeling Process 46

5.1.2 Editor Generation 48

5.1.2.1 Creating the Domain Generator model 49

5.1.2.2 Creating the Graphical Definition Model 51

5.1.2.3 Creating the Tooling Definition Model 51

5.1.2.4 Creating the Mapping Model 51

5.1.2.5 Creating the Editor Generator 52

5.1.3 Language Reprocessing 52
5.2 Semantics 52
5.2.1 Transformation Rules 54
5.2.2 Rule organization 58

6 Validation 59
6.1 First Set of tests 59
6.2 Case Study 60
6.2.1 ATLAS 61
6.2.2 Procedure 62
6.2.3 Results 65

7 Conclusion and Future Work 69
7.1 Conclusions 69
7.2 Future Work 70

A (H)ALL Metamodel Diagrams 71

B CO-OPN Metamodel Diagrams 83

C Editor Dependencies 89
C.1 Root Editor 90
C.1.1 Hall.gmfgen file variables 90
C.1.2 Hall.gmfmap file variables 91
C.2 UserProfile Editor 92
C.2.1 UserProfile.gmfgen file variables 92
C.2.2 UserProfile.genmodel file variables 93
C.2.3 UserProfile.gmfmap file variables 93

D ATL File Listing 95
D.1 Declarative01.atl 96

D.2 Declarative02.atl 98
D.3 Declarative03.atl 100
D.4 FSM01.atl 102
D.5 FSM02.atl 107
D.6 FSM03.atl 109
D.7 mfsm00.atl 111
D.8 mfsm01.atl 113
D.9 mfsm02.atl 118
D.10 mfsm03.atl 121
D.11 Data01.atl 123
D.12 Data02.atl 126
D.13 Data03.atl 130
D.14 Data04.atl 132
D.15 Data05.atl 135
D.16 Router01.atl 138
D.17 Router02.atl 143
D.18 Router03.atl 146
D.19 Router04.atl 153
D.20 Router05.atl 160
D.21 Router06.atl 164
D.22 Router07.atl 168

E XMI models used in testing 173

E.1 First Phase Test (H)ALL Source Model 174
E.2 Example of CO-OPN output 175
E.3 Case Study (H)ALL Source Model 177

List of Figures

1.1 Context of the presented work in terms of development flow 2
1.2 Problematics found in the 3

2.1 Model Driven Engineering layers within the context of domain language engineering 8
2.2 DSL development steps 9
2.3 Relation between transformation elements 11
2.4 BATIC3S methodology with highlighted thesis context 13
2.5 CO-OPN diagram view example of a money box module 16

3.1 Transformation languages feature model 21

4.1 Rule scope representation 28
4.2 Left hand source model instance 29
4.3 Right hand source model instance 29
4.4 Right hand source model instance set 29
4.5 Target model instance 30
4.6 Right hand instance and non-terminal symbol relation 30
4.7 Non-terminal symbol propagation to target instance 30
4.8 Non-terminal symbol propagation to source instance 31
4.9 Implicit instance and rule relation 31
4.10 Left and right side relation 32
4.11 Meta-model for the diagrammatic view of the transformations 33
4.12 Composed diagram of a rule 34
4.13 An ATL rule sample decorated with the defined vertices $V_{(_,_,_)}$ and function $f_{_}$ symbols, produced by the symbolic interpretation of the rule *sys2ctxuse*. 38
4.14 Layer chain 39

4.15 Incremental model approach 40

5.1 Root meta-model 46
5.2 Component inheritance meta-model 47
5.3 Component meta-model 48
5.4 PreConditionExpression meta-model 49
5.5 Inter editor dependencies 50
5.6 GMF Dashboard 51
5.7 (H)ALL editor interface 53
5.8 ATL rule example 55
5.9 ATL propagation rule 56
5.10 example of an ATL rule that requires the use of layers 57

6.1 general purpose User Interface for the Online Software of ATLAS 61
6.2 Specification of the Visual Components of a user's GUI 62
6.3 DAQ system and structure specification 63
6.4 FSM for component control 64
6.5 TreePanel XMI (H)ALL representation 65
6.6 TreePanel target context representation at layer Declarative02 65
6.7 TreePanel target data representation at layer Data01 66
6.8 TreePanel target context representation update at layer Data02 66
6.9 TreePanel target router construction at layer Router01 67

A.1 (H)ALL Meta-Model 73
A.2 Upper left section of the (H)ALL Meta-Model 74
A.3 Upper middle left section of the (H)ALL Meta-Model 75
A.4 Upper middle right section of the (H)ALL Meta-Model 76
A.5 Upper right section of the (H)ALL Meta-Model 77
A.6 Lower left section of the (H)ALL Meta-Model 78

A.7 Lower middle left section of the (H)ALL Meta-Model 79
A.8 Lower middle right section of the (H)ALL Meta-Model 80
A.9 Lower right section of the (H)ALL Meta-Model 81

B.1 CO-OPN Meta-model 85
B.2 CO-OPN ADT Module Meta-module 86
B.3 CO-OPN Class Module Meta-module 87
B.4 CO-OPN Context Module Meta-module 88

List of Tables

3.1 Meta-modeling tools analysis 20

3.2 Transformation tool analysis 23

1. Introduction

This thesis was made within the context of the Computer Science Master's Degree program held at Falculdade de Ciencias e Tecnologia of Universidade Nova de Lisboa.

1.1 Context

Within modern software engineering methods it is hard to conciliate requirements with specification and implementation of particularly large projects. To deal with this, new approaches have been created to tackle this problem. Some of these methods include the Model Driven Development (MDD) approach, that makes use of Domain Specific Languages(DSL) specification and Model Transformation techniques.

Model Driven Development sees models as artifacts in process where the specification of systems start at a more abstract layer and are transformed, in a automatic or semi-automatic manner, to more detailed specifications at a lower level of abstraction in a systematic way. The need for specifying the referred models at the different abstraction layers in some language motivate newly designed DSLs. Additionally Model Transformation is needed in order to give semantics to these models and carry them to new steps of the MDD approach. The major advantage of this structured approach is that the specification gap between design and implementation can be overcome, and it becomes possible, if properly designed, to derive solutions from a specification centered simply on the problem and not the solution space of the implementation.

The MDD approach can be used for the purpose of rapid prototyping with several different goals besides implementation. Simulation or Verification are also possible in a integrated way depending on the problem to be solved.

One domain, where we can see these issues in practice, is the one of complex control system.

In it the elevated number of elements and critical responses, lead to a difficult conciliation of requirements with implementation, the former usually taking precedence. The BATIC3S Project[1] was created to deal with these problematics. To do this an MDD approach was taken, as represented in figure 1.1, by introducing abstractions over the domain with a Graphic User Interface(GUI) specification language(in figure 1.1 referred as Domain Specific Modeling(DSM)), and carrying these specifications into a framework that introduced the abilities of producing testing, verification and implementation code capable of integration with the controlled system and a GUI engine.

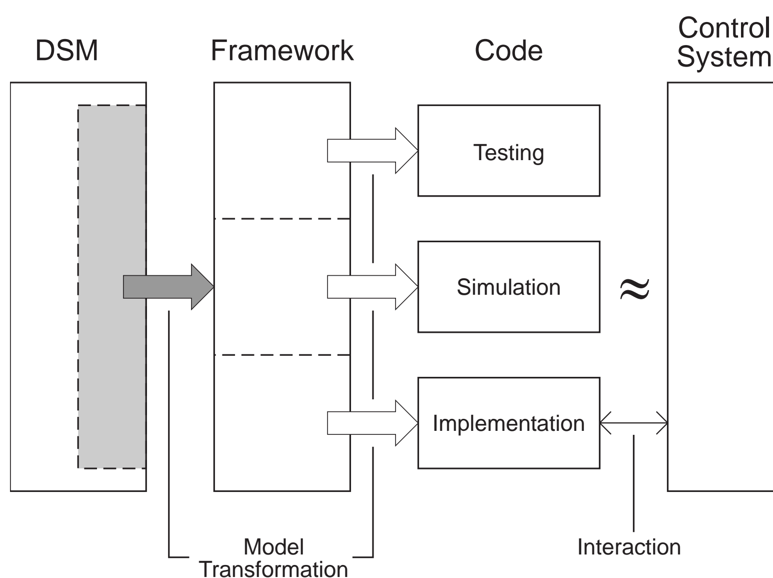

Figure 1.1 Context of the presented work in terms of development flow

[1] presented in section 2.3

1.2 Motivations

This thesis will focus in a particular portion of the BATIC3S project's development, specifically the implementation of a high level abstraction visual language named (H)ALL [2], in the form of a diagrammatic editor, and then tackle the gap between the high level models created with the editor and the target platform CO-OPN[3].

With the use of MDD methods, the need arises to show that the problem's domain in focus is correctly specified and expressed by these methods, and that the computational solutions provided do not suffer from the limitations, like error prone translations, caused by the semantic gap between the requirements specification in the models and their implementation when either the manual or automatic approach is used. With this in mind, we want to ensure that the models are, at the syntactic level, well-formed and that, at semantic level, the translation process did not introduce unwanted faults into the target models. Besides that we want to guarantee that the obtained model is syntactically correct according to the target language's syntax, in our case the CO-OPN language. The mentioned problems of each step are illustrated in figure 1.2.

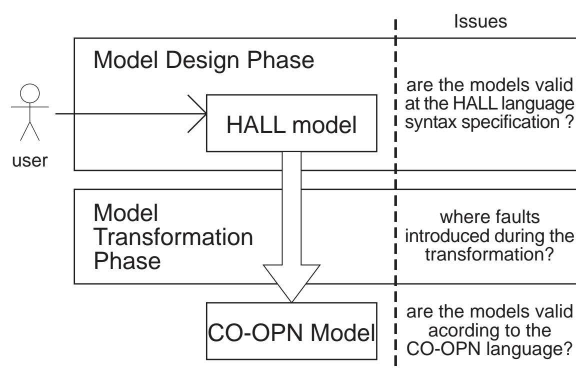

Figure 1.2 Problematics found in the

To achieve this, we need to study processes that allow us to check these methods limitations. And so, with this thesis we will implement the aforementioned language of the BATIC3S project

[2]presented in section 2.3.1

[3]presented in section 2.3.2

in a first phase and then verify, as explained, its integrity to find any flaws that may exist on the processes involved, producing correct results.

1.3 Proposed Solution

In [2] we have studies on the approach to control systems, its GUI specifications and shortcomings. In the sequence of that study, it was created a language named (H)ALL, suited to tackle the observed shortcomings. From the initial specification, we move on to the implementation of a editor, allowing us to further develop the language and eventually redesign it, due to its own shortcomings. One of the possible reasons for that redesign is that the proposed language is visual and domain specific, which makes it highly dependent on its usability, and not only on the specification of concepts.

Having this in mind we establish the use of automated visual editor tools instead of building one from scratch. This enables a faster development of the editor, and a greater feedback to the language development without compromising the already produced work. Besides that, the use of these tools, give us guaranties of the model's syntactic correctness.

From the created visual editor, *we* have the possibility of creating models that will be translated to the target platform. To achieve this, like for the editor creation process, we look into existing tools for model transformation, and search for the best suited for this task. After we select a tool that copes with the needs of this particular project, we proceed with the specification of the transformation rules, but first we design a transformation procedure , that enables us to assume that after following it during the transformation rules design, the models produced by them are correct.

As the rules are created and we can generate partial models in the target platform, we test them for syntactic correctness, and an observational verification of the semantic correctness of the transformation.

To validate the approach we then make use of a real life case study.

1.4 Document Structure

In this chapter, we introduced the context, motivations and a proposed solution to the presented problem.

In chapter 2, we present a series of preliminary concepts and related work, so that people not familiar with the context of this thesis can better understand the context, the problem and the solution presented in this thesis.

In chapter 3, we describe related work, giving an insight to the *evaluation* methods and tools associated to the proposed solution. First, we analyze the elements associated with the meta-modeling of the high level elements of the solution and the editor implementation. Afterwards, we evaluate tools and methods that allow the suppression of the gap between the design models and the target platform solution.

In chapter 4, we focus on the syntactic evaluation of the produced work and the properties and guaranties we can derive from this evaluation, and we analyze the semantic properties of our solution and the way we can ensure that any semantic present at design time will stay unaffected throughout the proposed process.

In chapter 5, we begin to describe the process of engineering that lead to the creation of the language editor and the transformation process that transfers the created models into the target framework. In this chapter we emphasise key aspects of development and implementation, as to facilitate the recreation of the elements produced in this thesis.

In chapter 6, observational testing and case study is described, and the way this evaluation adds to the previous verifications of the solution.

In chapter 7, the conclusions of the developed work are presented. In it we describe the achieved goals and highlight the technological limitations. Finally we present the course of future development and the way we can improve the achieved results.

After the conclusions, a section of Appendixes is present. In these we present the meta-models used for our solution. The variable setting for the creation and chaining of the model

editors. The listing of the produced transformation rules and layers. The listing of synthetic and specific test models examples.

2 . State of the Art

To understand the proposed solution, we must take an understanding in a series of methods and tools, which are presented in this chapter. We will start with general and move on to more specific, project centered concepts along the chapter.

2.1 Model Driven Development

Model Driven Development(MDD), also described in the literature as Model Drivel Engineering (MDE) [11] [33], has its focus in the isolation of the problem's context from its implementation aspects. To achieve this, a series of mechanisms are raised, to allow the representation of a problem's solution in a abstract manner, through the use of models. This type of abstraction can be achieved by using the problem's domain own terms when defining the solution's models. Typically it is made use of Domain Specific Languages to capture the domain terms and their syntactic definition. All this allows that the domain expert can focus his efforts on th problem's solution, without any concern with details that do not contribute to the improvement of said solution, meaning how it is implemented in the solution domain(in programing steps). This type of approach is visible in figure 2.1, where we can observe the relation between the several abstraction layers and how this can create an abstraction of the implementation process. It shows the meta-meta-modeling layer(where we have the language used to specify other languages), the meta-model layer(where the description of the domain language rules is made) with an example of a language specification meta-model, the modeling layer(where models are defined with the use of the specified DSL) with an example of a model instance, and finally its implementation in code. Further advantages of this is the capability of developing solutions with multiple implementations(ex: multi-platform deployment) and any improvements on the implementation do not interfere with the solution and vice versa. We will now define this concepts in more detailed manner in the following subsection.

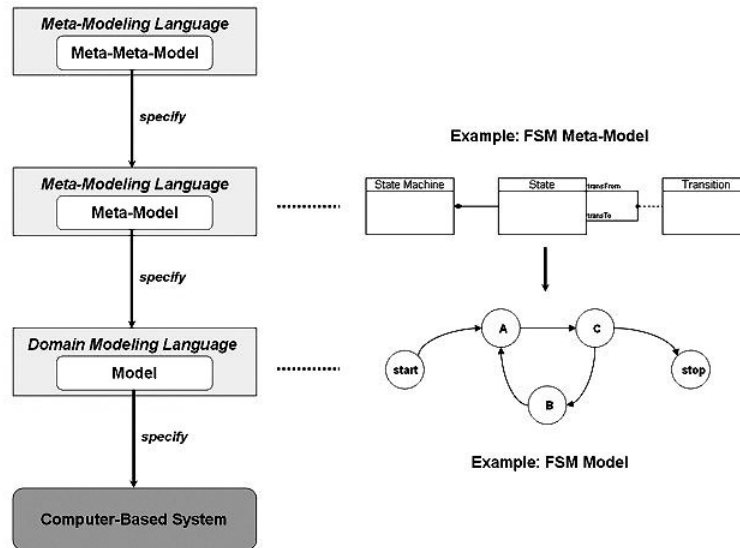

Figure 2.1 Model Driven Engineering layers within the context of domain language engineering from [8]

2.1.1 Meta-modeling

In face of recent technologic developments, the definition of meta-model and its use, has become vague and it spreads through development areas that go from computer sciences to industry. Still we can establish meta-modeling in the context of this thesis, as a rule definition and abstraction mechanism, that allow the specification models and terms, from particular contexts. This facilitates the understanding of problems and helps coordinate partial solutions.

The concept of meta-modeling, is tied to the type of solutions being produced, imposing to the created models, abstract characteristics relevant to the consistency of the produced solutions. So when we talk about meta-modeling in the context of DSL's, we are dealing with the rules that define that language's abstract characteristics, derived from the analysis of the development domain.

2.1.2 Domain Specific Languages

The concept of Domain Specific Languages(DSL) is tied to that of abstraction. Through a specification paradigm, such as meta-modeling or language grammars, a set of domain concepts capable of expressing solutions for that domain are defined. This set of domain concepts allow the abstraction of everything outside of the domain of development, such as specific implementation details, focusing solutions on solving the problem.

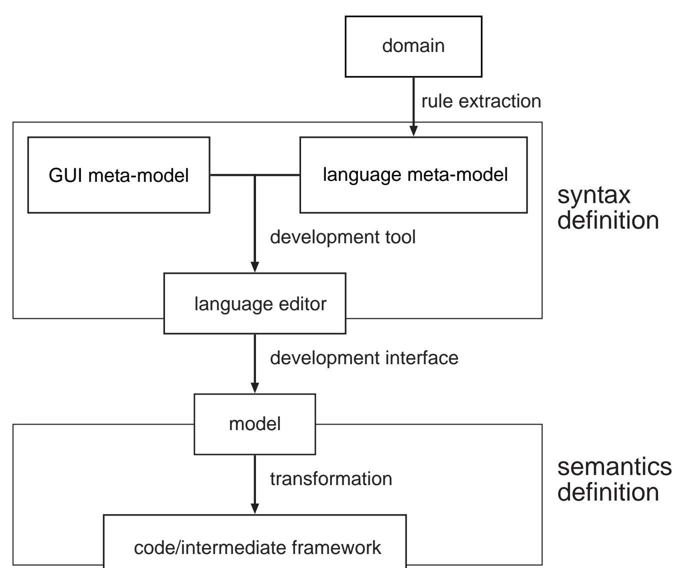

Figure 2.2 DSL development steps

As viewed in figure 2.2, the development process of a DSL, starts with the analysis of the target domain, and the collecting of abstract terms tied to that domain. These terms are then expressed in a formal manner. For Visual DSL's, this can be done with the help of a DSL specification tool. Most modern specification tools, use meta modeling paradigms such as UML and entity relation diagrams, to specify visual languages, as a way to both connect to used enterprise procedures and because it simplifies the refactoring of the language as it is being developed in collaboration with target domain experts. With the formal elements defined, the visual definition of these elements, through the use of a GUI meta-model, and the interaction

process with the user are defined. With the combination of these two sets of specifications, the language terms and the visual and interaction definitions, it is possible to produce an editor for the created language. In most modern tools this step is produced automatically. With these steps we define the syntax of the DSL, this will allow the creation of models that can be used to refine and verify the specification itself or to produce other forms of output such as execution code or documentation. The transformation of these models into other specifications, sets the definition of semantics for the DSL terms.

2.1.3 Model Transformation

Model Transformation is the process by which models compliant to a set of rules, are turned into models compliant to a new set of rules[22, 21, 4]. In the context of this thesis these rules are defined by means of meta-models. The transformation process itself is guided by a set of transformation rules that specify for the starting patterns what needs to be changed so that they can cope to the target meta-model. The relation between the elements involved in this process can be observed in figure 2.3, where the source, target and transformation rules conform to their respective meta-models, and all of these share a common meta-meta-modeling specification paradigm. This makes it possible to understand the terms used for each model and proceed with the transformation accordingly.

This type of process can be as flexible and free as we want it to be. For this reason it becomes necessary to ensure that when the transformation rules are created, they will produce the correct results. For this reason in this thesis we will approach this specific situation and study manners to verify this process.

The concrete applications of model transformation are such as:

- Obtaining different "views" over the same model
- To iterate optimizations within the same model, in the case of both source and target meta-models are the same. This can produce optimized solutions that otherwise could

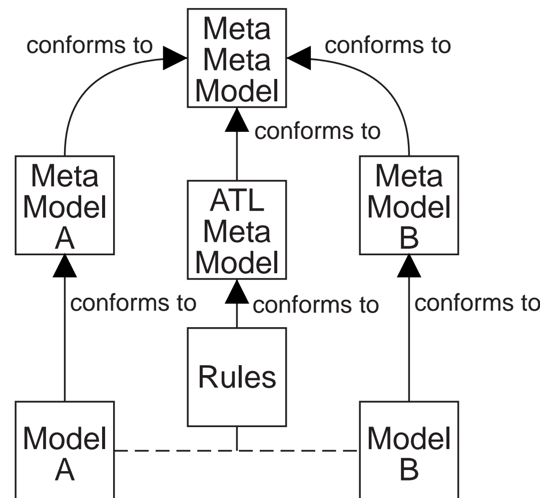

Figure 2.3 Relation between transformation elements

not we obvious or wold take mush work(ex: balancing a tree data structure)

- Adapt a solution model to different frameworks, abstracting from the process the differences of the different systems.
- To generate textual models, including automatic production of documentation.
- Application of model composition technics such as Model Weaving.
- Ultimately we can consider an execution step as a model transformation, were the source and target meta-models are the same, and its successive application leads to an algorithmic equivalent result.

2.1.3.1 Graph Transformation

Graph transformation[21, 7] is a very well studied field, and can be observed as a particular case of model transformations, were models conform to the abstract definition of graphs. This is a very important step in the validation of the proposed solution, due to its high level of study and the studied characteristics such as coverage and structural equivalence of source and

target models. As such, it is essential to understand the concepts used in both graph description and graph transformation, so that one can observe the parallels created in the solution and understand them.

We will describe transformation process in more detail in the following chapter, when describing the correctness of our approach.

2.2 Control Systems

Control systems are systems composed by one or more control elements, whose objective is to observe, manage and control other elements of a systems that can also be the control system itself. The type of system being monitored by a control system can be as varied as logic and linear systems, with analog or digital elements, and deal with discrete or continuous events, or any combination of these situations.

The control systems focused in this thesis, can be classified as centralized reactive systems to observe and control critical, hierarchically structured systems, with an elevated number of elements.This means that usually extremely specific formation is necessary to operate and maintain such systems.

2.3 BATIC3S

The BATIC3S(Building Adaptive Three-dimensional Interfaces for Critical Complex Control) project[30], was initiated in 2005, in a collaboration of Université de Genève(CH), Ecole d'Ingénieur de Genève(CH) and Universidade Nova de Lisboa: Faculdade de Ciências e Tecnologia(PT). Its general goal is to create and develop methods and tools that allow for the rapid prototyping of adaptive 3D user interfaces. It also tackles the problem of the subsequent semi-automatic generation of the referred UI implementation, testing and verification, in the domain of large

scale complex control systems.

To achieve this, an MDD approach was taken for the definition and creation of the interfaces, where the overall process, is also managed with an abstraction philosophy over the remaining aspects of implementation as shown in the BATIC3S methodology diagram in figure 2.4. In it we can observe, on the top semi-circle, the use of several abstractions over the specification of a control system, such as *user profiles*, *system structure*, *system static behavior*. These abstractions are then combined into a single model. This combination is no longer under the control of the domain expert that specifies the control system, and is made into the CO-OPN framework, as to allow intermediate verification of the models, production of simulation code, and the generation of implementation code that will interact with the concrete system and a 3D GUI interaction engine.

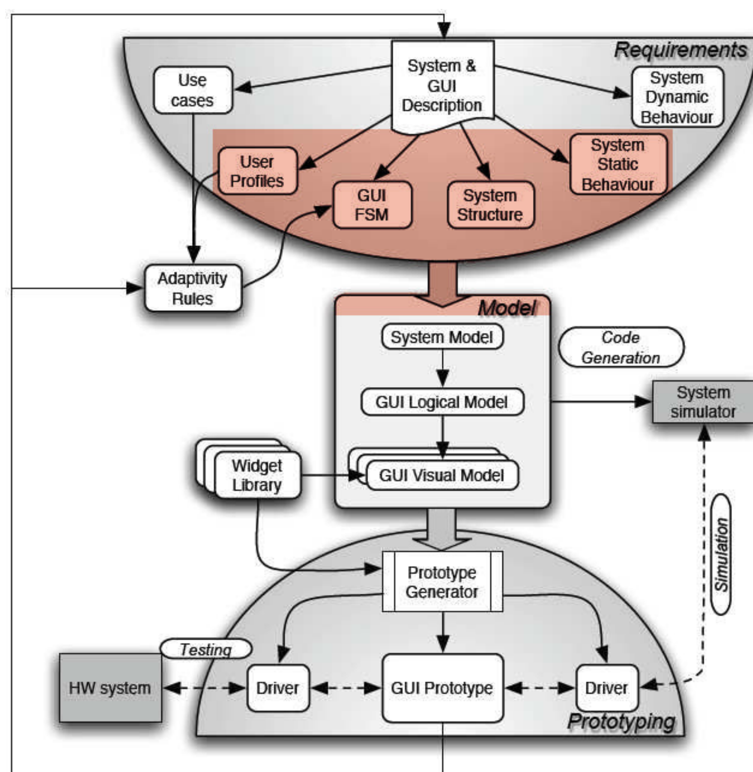

Figure 2.4 BATIC3S methodology with highlighted thesis context

For first steps of the definition of the graphic user interfaces, this was achieved within the project, by specifying a DSL for this purpose, as to define them in an abstract manner, and without any reference to the concrete implementation. The connection of the abstract specification with the implementation section of the BATIC3S project is made through model transformation into the CO-OPN framework. These two steps are part of the subject of this thesis, and highlighted in figure 2.4.

2.3.1 (Human) Assisted Logic Language

The (Human) Assisted Logic Language((H)ALL), specified in [2] and [1], is a domain specific visual language, that targets the specification and prototyping of large scale complex control systems graphic user interfaces. These objectives are achieved through a segmentation of interface specifications into a hierarchic definition of components and users. These elements behavior is then defined through the use of finite state machines, where the triggering of transitions and the values of states are defined with algebraic expressions. The purpose of this separation is to offer a more intuitive view of each aspect of the development of a controll system interface. The interaction between the specified elements of the interface, is made through a system of message propagation across the hierarchy.

The original specification of (H)ALL, designs its semantics using a set of model transformation rules in Queries/Views/Transformations (QVT) [31]. Due to an absence of implementation of QVT, a new transformation process capable of setting the semantics of the (H)ALL models has been studied and selected during the course of this thesis.

2.3.2 CO-OPN

The CO-OPN(Concurrent Object-Oriented Petri Nets)framework is based on algebraic description of Abstract Data Types(ADT), and the description of models through a formalism based on algebraic Petri Nets to represent behavior and concurrency, such as synchronization of events,

where all these aspects are encapsulated in a Object-oriented paradigm. The operational semantics present in CO-OPN, makes this framework open to rapid prototyping and simulation of the models. A coordination layer confers expressiveness to model design, providing a form of representing distributed computation in an abstract way by representing the interaction between the modeled entities. Furthermore, the framework's syntax and semantics, allows the use of object-oriented heterogeneous concepts.

Model specification is made through a collection of ADT's, classes and contexts in the CO-OPN modules, and in a abstract and axiomatic form. This type of specification makes CO-OPN effective as a model transformation target, of DSL models, because by using a modular construction, it allows the specification of the several components of a DSL, and their relations.

The development platform for the CO-OPN framework(CO-OPN builder [18]), allows the semi-automatic generation of code prototypes, execution and simulation of the prototypes, the enriching of the prototypes with customized code, test application and prototype execution for validation[26]. CO-OPN builder also allows the generation of diagrammatic representation of models, like that of figure 2.5, from their textual specification. In this diagram, we see an example of a model of a money box, with the use of places(circular elements inside the money box module) to store values, gates to provide the interaction of the money box with other modules, and methods that connect these several elements, and order the flow of data inside the module through the use of algebraic expressions.

Further information on CO-OPN can be found in [5], [29] and [27]

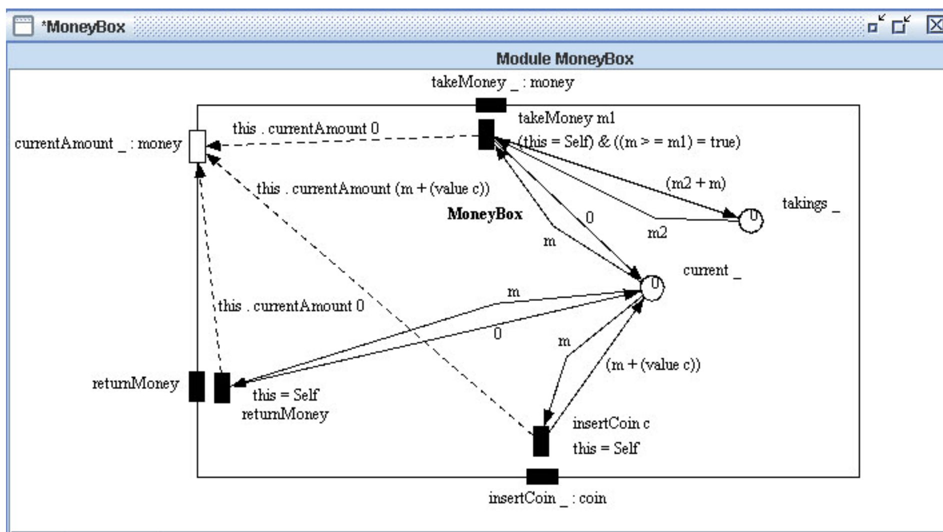

Figure 2.5 CO-OPN diagram view example of a money box module from [2]

3. Tools

In this chapter, we present an analysis of tools and methods used to model a domain language. We will begin by looking at tools that allow for the specification of visual domain specific languages(with the use of some meta-modeling formalism), and the automatic creation of an editor for those languages. From the models creatable by these editors, we will observe their expressiveness and compatibility with further steps of the development. These steps involve the transformation of these models into models in the target framework, which confers semantics and allows to have a set of operational tool for testing, execution and simulation. To do this we need to look at tools and methods that allow this transformation, and also allow the verification of the transformation process. From these observation, we will select the most suited to achieve the proposed solution.

3.1 Meta-Modeling Tools

Although being possible to use other processes, such as graph grammars[], most recent approaches to visual domain specific languages, is done by specifying a meta-model of the created language. This approach means we specify the language through elements and connections between those elements. It also means that any combination of element connections is valid, and in some cases, this is not desired in the language specification. For these situations, restrictions are added to the language specification, either through OCL expressions or through compilable code, and could be seen as a form of type checking(also called static semantics).

The referred approaches, allow for the automatic generation of visual editors, that can verify the syntax of the visual language at modeling time. This automatic procedure, facilitates both the use and creation of such editors, through a more interactive process, therefore, they are suited for our requirements.

In benefit of this approach we start by choosing the best fitted tool for a solution. For the

selection of the meta-modeling tool, we looked at [32]. From these we evaluated the presented tools and characteristics. From the information collected in this manner, and based on the requirements of our solution, we established analysis criteria and candidate tools for meta-model and editor generation process. The final candidates were then evaluated for usability through experimental use. In this way, we try to establish, an objective decision in our choice of the meta-modelling tools.

3.1.1 Meta-Modeling Tools Decision Criteria

So based on the existing evaluations and the requirements of our solution, we use the following criteria:

- can express all the **expressiveness** of (H)ALL, i.e. it is capable of representing all syntactic requirements of the language, as to produce a correct metaphor of the domain.
- can express its **restrictions**, either by means of meta-model expressiveness or through the use of additional specifications, like the use of OCL or compilable code.
- can **generate** a modeling interface that meets the expectations of the target modelers, i.e. it will run in the development environment already used by the modelers, and that it possesses an interface that is intuitive or of rapid learning to those same modelers.
- Created models are suited to undergo the **transformation process** into the target framework, i.e. that models are already in a standard format or that format can be expressed to the transformation language.

3.1.2 Meta-Modeling Tool Analysis

From all the observed tools, only GME [8], DSLTools [28] and GMF [10] show themselves to be the most prominent and best suited for this task, because they were capable of producing a

complete and integrated development process. So now we will proceed with a closer evaluation of these tools in the rest of this section.

GME has proven to be the easiest to use of the three, capable of expressing (H)ALL's requirements and restrictions, using meta-modeling process close to MOF. Nevertheless, the models created by this tool although in XML, were based in an unpublished specification discouraging its use with other tools outside of the GME environment. The interface of the generated editor although limited, presented a good level of usability, providing customizable element features, the separation of tools into tabs for better organization and providing command line interaction.

DSL Tools required the most adaptation of the original (H)ALL proposal to its own meta-model paradigm, which is not based on the MOF standard, unlike the other selected tools. Additionally, no constraints expressiveness was provided by this tool at this point. The interface although being somewhat rigid, allows for the inclusion of customizable elements, and the organization of a Toolbox. The models generated by DSL Tools are defined in a proprietary format of XML, discouraging its use outside the development tool's environment.

GMF has its models defined in the eclipse *ecore* format. This format is based on the MOF standard, and is paired with a set of tools that allow its exchange with other formats. During the editor specification process, it is possible to add OCL and Java code constraints to its use, but not directly into the meta-model. The interface, although hard to specify beyond the default, allows for its full customization. The models generated by this tool abide by the XMI standard, allowing full integration with any tool that is compliant with this standard.

As a consequence of our analysis summarised in table 3.1 we chose GMF as our meta-modeling tool to implement our solution. We chose it based on its capabilities of expression, standards use, evolution potential and capability to interact with other tools. Additionally other tools of the BATIC^{3}S project are also converging to the same development environment, the eclipse platform.

Name	DSL Tools	MetaEdit+	**GMF**	GME	AtoM^3
Expressiveness	+	+	**+**	+	-
Standard Meta-Model Definition	-	- -	**++**	++	+
Resulting Interface	+	-	**++**	+	-
Restrictions Definition	-	N/A	**++**	+	+
Cross Platform	-	+	**+**	-	+

Table 3.1 Meta-modeling tools analysis

3.2 Transformation Tools and Processes

Although defined as a standard, to be added to the also standards MetaObject Facility(MOF)[14] and Unified Modeling Language(UML)[17] specifications, by the Object Management Group(OMG)[16], QVT[31] still has no tools that fully implement its use. Nevertheless from their efforts to build this standard, many transformation languages have risen with solutions that complement it with alternative methods and specifications, such as VIATRA[6], while others tried to integrate some level of compatibility with QVT in themselves[13], such as the Atlas Transformation language(ATL).

Not being possible to use QVT as our transformation language, we need to select tools and methods, that allows for the correct transformation of the created models into models of the target platform and that allow for the *validation and* verification of this process. To do this, we started by looking into a survey [22] that clarifies the different model transformation aproaches. We also looked at a survey to evaluate model transformation processes[21], such as graph transformation, triple graph transformation, relational algebras, XML to XML transformation by evaluation of it's text patterns and even mixed solutions, their features and tools that support them.

We first study the classification of model transformations. This is necessary, so that we can

select the methods that are better suited for this particular transformation process. This benefits the selection with a greater control and understanding of the transformation process, also essential to the verification of this step, by allowing us the association of the methods used in transformations, to formalisms that can help us to verify those transformation. Additionally we look at a set of features present in the evaluated transformation tools. These features, presented in figure 3.1(where black dots mean mandatory features and empty dots mean alternative), as referred to in [22],are such as, bi-directionality of the rules, if the transformation is typed, if there is a need to use intermediate structures or models for the correct execution of the transformation, if method application is deterministic, a full application is destructive to the source model and if it is possible to define or other sets to the rules as to ensure integrity and convergence of solutions.

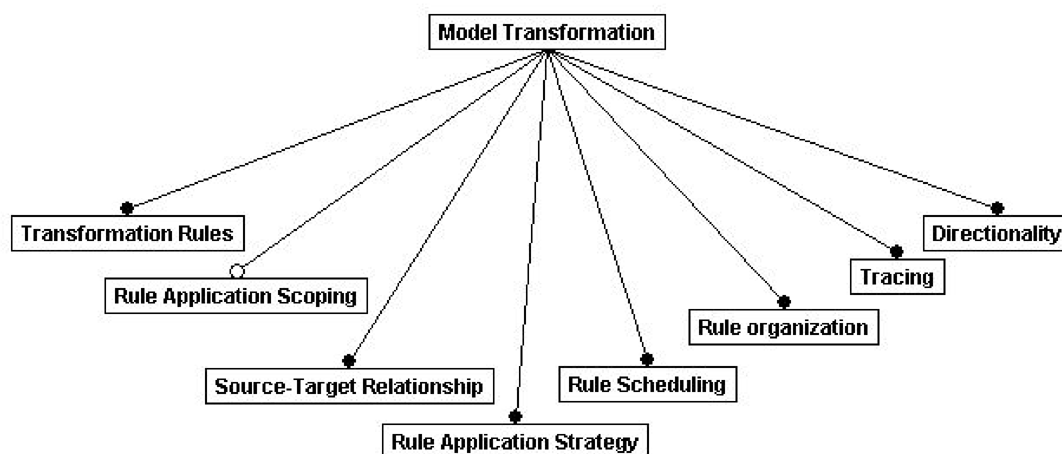

Figure 3.1 Transformation languages feature model

from [22]

From the analysis presented in [22] and [21], we are also shown that through compliance of graph transformation to the source and target meta-models, similar to that present in figure 2.3, syntactic validity of the transformation is guaranteed. In [12] is shown that this type compliance is also present in ATL.

3.2.1 Transformation Tools Decision Criteria

So we establish our needs from a transformation process and language in the following manner:

- it has to allow *a formal* **verification**, ether by itself and its implementation, or by allowing its comparison with known formalism.

- it has to be capable of detecting and generating the desired model patterns

- it has to ensure that created models **conform to** their **meta-model** specification

3.2.2 Transformation Process Selection

From this evaluation, as presented in table 3.2, Viatra2 and ATL were selected for a further analysis. Testing the two languages for their concrete capabilities.

Viatra2 although promising in use of graph transformations, its use of state machines to control the transformation process, dictating the sequence of the transformation rules, and its compatibility with the environment and models of the model generating editor, it did not achieve the usability necessary to implement our solution.

ATL showed to have a good level of expressiveness, allowing the declarative and imperative specification in the transformation rules. Furthermore, ATL allowed its comparison with the graph transformation formalism, although not used for the transformation process itself. Integration with the chosen development, the Eclipse framework, support and standard status within the model to model Eclipse development project and compatibility with the editor generated models, added to the choice of using ATL as our solution transformation tool.

	ATL	Viatra	QVT	GREAT
Visual Rule mapping	-	+	-	+
Multiple Meta-Models	+	+	+	-
Declarative Rules	+		+	
XMI compatible	+	+	N/A	-
Usability	++	-	N/A	+
Syntactic Conformity	+	+		+

Table 3.2 Transformation tool analysis

3.3 Summary

In this section, we have proceeded with a deep analysis, where the selection of the GMF/EMF language workbench and the ATL transformation tool were justified. We have also observed that these type of tools are rapidly evolving. Although the Meta-Editor tools present some limitations of expressiveness, they are benefiting from the paradigm they implement, allowing the adoption of MDD methods in developing the tools themselves. This allows for a rapid development cycle, that unfolds in their rapid evolution allowing for better expressiveness, usability and stability. Knowing this, we are aware that the chosen tool for this task will be outdated in a near future, but most of the core procedure will be maintained. Besides that, these tools facilitate the edition and generation process, becoming easier to update and maintain our developments. As for the transformation languages, many are being added visual editors, although not yet usable with ATL at the time of our thesis, allowing the representation of rules and patterns at the same level as the model creation. The transformation processes are also evolving in terms of efficiency, becoming faster and more reliable, and also in terms of expressiveness, allowing for the description of more complex transformation patterns.

Concluding, it is expected that many of the shortcomings of these tools and methods become solved in the near future, and with the evolution of code generation techniques, we foresee that

these type of tools will become fully integrated development tools, as already occurs today with general purpose textual programing languages.

4 . Correction of the Approach

In this chapter, we study the approach and methods used to implement the (H)ALL language and deliver it to the CO-OPN framework.

This study is made in two phases. In the first phase, we study the syntactic properties and guaranties given, by the meta-modeling tools, in order to understand if they are already being properly covered. In a second phase, we propose a set of properly formalized procedure, that ensure the semantic correctness of the transformation rules.

4.1 Model Syntax Evaluation

To guaranty the soundness of the development process, and that no errors are introduced in the models, we must first evaluate the syntactic guaranties of the tools used. First we observe the Meta-modeling tools to ensure that the user describes a well-formed initial model. Following that, we evaluate the transformation process itself for the maintenance of that well-formedness.

4.1.1 Meta-Modeling Tools

While editing models in GMF, these are maintained in conformity the their specification meta-model, as the editor, does not allow the creation of patterns not present in meta-model. The syntax of the models can be further restricted with the use of constraint rules.

Although useful in the current implementation of the (H)ALL meta-model, for detection of possible circular referencing in the hierarchy constructions, these suggested rules where not created. This occurred, because the particular rules necessary, implied the use of non standard OCL[15] rules, or the inclusion of non generated Java code into the editor, and this wold fall outside the objectives of this thesis.

Another way to cope with these syntactic limitations, could be made through detection

of the fault patterns in the transformation process. But this implied the design of an error detection meta-model, and it compromised the termination guaranties, present in the application of layered rules.

Furthermore, these limitations of the current version of (H)ALL, are not assured in new implementations of (H)ALL, as the language and its interaction are still evolving.

For these reasons, and the still restricted use of the (H)ALL editor, we can assume the syntactic correctness of the (H)ALL models.

4.1.2 Transformation Tools and Processes

With the study of model transformations we observe that there is a conformity relation between models and their respective meta-models. So rules set to patterns present in those meta-models, produce compliant models, if the source models are themselves compliant to their respective meta-models. This is shown in [21], and illustrated in figure 2.3. If such compliance is not met the transformation tools are not capable of recognizing the rule application patterns, invalidating the full transformation process.

Taking this in consideration we can assure that, the given well formed (H)ALL models will produce syntactically correct CO-OPN models.

4.2 Formalizing the Transformation

As there is no formal description of the ATL, one is described in this section, in the form of rule application used in our particular transformation process.

We only evaluate a small part of the ATL language, otherwise it would fall beyond the scope of our objectives, and a very time consuming task to fit in our master thesis scheduling. Working with only a portion of ATL, we chose the declarative approach, because it allowed for a more direct parallel to graph transformation and a more transparent formalization process.

4.2.1 Formalisation attempts over model transformation rules

There have been some research in the formalisation of both model transformation rules and its validation properties. However, we question the level of abstraction of these formalisations, since they typically do not give us (in general) enough rationale in order to be able to decide about the satisfaction of some validation properties in a particular set of model transformation rules, in a systematic way.
In the work of [25], it was presented a clear separation from a specification of a model transformation and its implementation. A formal language for the specification of model transformations (MTSpecL) is presented to allow a pure specification of model transformations with the notion of contracts, and regardless of its implementation. However, in this particular case, this formalisation is applied to the problem of generating test cases for model transformations expressed in several kinds of transformation languages (ATL, etc.). Furthermore, this definition seems to be closer to the model transformation designer, in terms of usability and prototyping, instead of achieving a clear rationale to perform verification and validation over the transformation rules.
In [34], it was presented a definition of the property of metamodel coverage, which includes feature, inheritance, association and model elements. Furthermore, it was presented algorithms to measure the metamodel coverage over the defined transformation rules, which is an important step to perform verification over transformation rules. In this work, they have chosen to experiment the Tefkat transformation language, but they did not present any kind of formalisation over this language.

4.2.2 Partial order isolation of the transformation rules

There also have been several studies which indicate a significant effort in the partial order isolation of transformation definitions in model transformation languages [23]. In [24], it was described an interesting idea of organising rules in layers, and presented as an important feature

in the life cycle of model transformation designs. Also, it was presented some important guidelines and properties that need to be checked during the validation of some model transformation design like:

- Syntactic correctness of both input and output models.
- Termination and confluence (unique results and determinism).
- Semantic equivalence or semantics preservation.
- Safety or liveness (to ensure preservation of structural or security properties).

4.2.3 Auxiliary Diagrammatic Formalism

First we derive the rules in a schematic form, with the main characteristics needed to help us evaluate the scope and repercussions of each rule application, and to define the ATL declarative sub-set we will be evaluating.

For the characteristics evaluated in the schematic diagrams, we create the following symbolic representation. Rule scope is presented as in figure 4.1. Source and target model instantiation , represented by □ and ○ respectively. Non terminal symbols, represented by ◇ . And propagation is made explicit with ⟶ . This last encompasses instantiation of non terminal symbols, rule relation and left and right element relations.

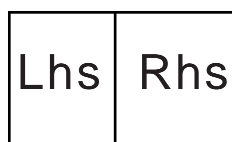

Figure 4.1 Rule scope representation

The rule scope is characterised by a right hand side corresponding to the **from** part of the ATL rule, and a right hand side corresponding to the **to** part of the ATL code. Furthermore left hand side declarations can be used on the right hand side and right and side can only be used

on the right hand side, but regardless of declaration order. In our use of ATL the declaration's scope is limited to the declaring rule itself.

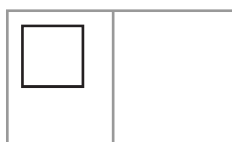

Figure 4.2 Left hand source model instance

Figure 4.3 Right hand source model instance

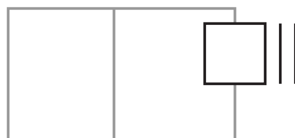

Figure 4.4 Right hand source model instance set

The source model instances can be represented on the left side of a rule scope(figure 4.2), making them as part of the rule's application pattern, or on the right side of the rule. When it appears on the right side(figure 4.3 and 4.4), it always represents a potential instantiation that relates to another rule application, and can be a single application(□) or a set(□‖). In the ATL code this appears as an implicit declaration through a relation or directly referencing an element of the source model. This potential state is made explicit in the schematic form by placing the instance in the rule's right outer boundary.

Target model instances only appear on the right hand side of the rule scope(figure 4.5), and terminate the rule propagation flux. The continuation of this flux is made through non terminal symbols within the target model instance. These refer to attributes that reference or aggregate

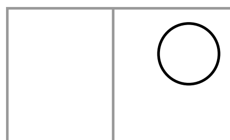

Figure 4.5 Target model instance

other instances. This close relation between the target instances and non-terminal symbols is represented placing the non terminal symbol directly below the related instance(figure 4.6).

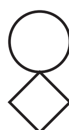

Figure 4.6 Right hand instance and non-terminal symbol relation

The non-terminal symbols can then propagate the rule to other terminal target model symbols, creating a chain of instantiations within a rule(figure 4.7), or propagate to a source model reference(figure 4.8), postponing the propagation to another rule.

Figure 4.7 Non-terminal symbol propagation to target instance

The propagation, although viewed with a unique arrow symbol, it conveys three different meanings, but all of them contributing to the rule application in sequence. The non terminal symbol reference to an instantiation(figures 4.7 and 4.8), for every time a non terminal symbol is raised, an instantiation, be it implicit or explicit, must be associated to it. A parallel between implicit instances and the rule that will explicit that instance(figure 4.9). And the internal association between the left hand side pattern elements and the right hand side(figure 4.10).

From the diagram elements specified we build a reference meta-model(figure 4.11, that allows us to build diagrams parallel to the rules in the manner presented in figure 4.12.

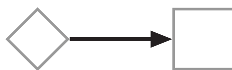

Figure 4.8 Non-terminal symbol propagation to source instance

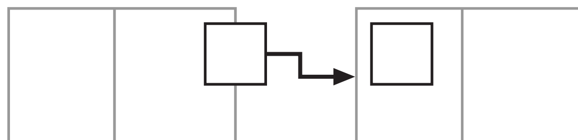

Figure 4.9 Implicit instance and rule relation

4.2.4 Base Formalization Concepts

With the rule analysis aid specified, we now set the basic concepts in the formalization process. For this we can look at a model as a set of vertices and edges in a directed graph description of the model. But, as this is a representation of a DSL model, just the vertices and edges are not enough. These items, must also be labeled to distinguish their properties as domain specific elements.

Also as we make the transformation process through ATL, the origin models and the target models are strictly separated. This separation is represented as Left hand side and Right hand side models. Where we only read from the Left hand side, and only wright on the Right hand side.

Because of this strict separation and rule sequence needs, we chose to layer the set of transformation rules, making it possible to read the output model of a given layer as an input of any subsequent layer, usually the next layer.

Having the notion of vertices, edges, labeling, left hand side, right hand side and layers, we begin do describe the formalization as such.

Let Σ_{VL} and Σ_{EL} be the finite alphabets that label the vertices and edges of the left hand side models, and Σ_{VR} and Σ_{ER} the finite alphabets that label the vertices and edges of the right hand side models. The sets of vertices are V_L for the left hand side vertices and V_R for the right hand

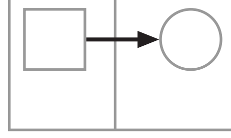

Figure 4.10 Left and right side relation

side vertices, where $V_L \cap V_R = \emptyset$. In the same manner the sets of edges are E_L for the left hand side edges and E_R for the right hand side edges. Where E_L is composed of left hand vertices and edge labels in the form $E_L \subseteq V_L \times \Sigma_{EL} \times V_L$ and E_R is composed of right hand vertices and edge labels in the form $E_R \subseteq V_R \times \Sigma_{ER} \times V_R$. Finally let $l_L : V_L \rightarrow \Sigma_{VL}$ be a total function that associates left hand side vertices with left hand side vertex labels and $l_R : V_R \rightarrow \Sigma_{VR}$ a total function that associates right hand side vertices with right hand side vertex labels. The triples

$$g_L(V_L, E_L, l_L)$$

and

$$g_R(V_R, E_R, l_R)$$

are directed labeled graphs of the left hand model and the right hand model, over Σ_{VL}, Σ_{EL} and Σ_{VR}, Σ_{ER} respectively or just left hand graph and right hand graph. These notions are present for each individual layer, but because more concepts are needed for the correct definition of the layering process, they will be presented later in this section.

We now add the notion of subgraph, where given two graphs g_i and g_j, g_i is a subgraph of g_j, in symbols $g_i \subseteq g_j$ iff

$$V_i \subseteq V_j, E_i \subseteq E_j, l_i = l_j | V_i$$

We also add the notion of incident edges, where g_L incident to $v_L \in V_L$ is

$$inc_{gL}(v_L) = \{(s_L, el_L, t_L) \in E_L | s_L = v_L \vee t_L = v_L\}.$$

If we consider $W_L \subseteq V_L$ a non-empty subset of V_L, then

$$inc_{gL}(W_L) = \bigcup_{v_L \in W_L} inc_{gL}(v_L)$$

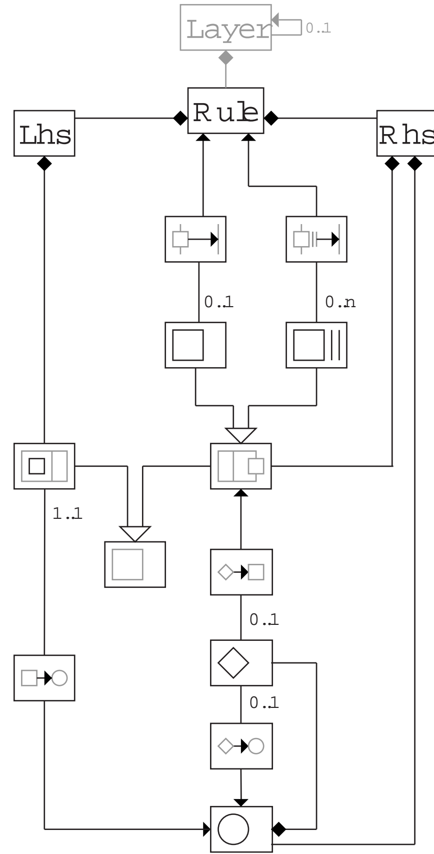

Figure 4.11 Meta-model for the diagrammatic view of the transformations

Finally we add the notion of path where a direct path between two nodes is defined as the existence of an arch that directly connect those two node, which can be formalised as: for a pair of nodes (A,B) there is a direct path $path(A,B)$ if

$$inc(A) \cap inc(B) \neq \emptyset$$

For a broader definition of path between A and B we take that if there is a path between A and C and there is a path between C and B there is also a path between A and B. This can be written as

$$path(A,C) \wedge path(C,B) \Rightarrow path(A,B)$$

where this notion can be applied to any sub-path, until it can be resolved as a direct path. These

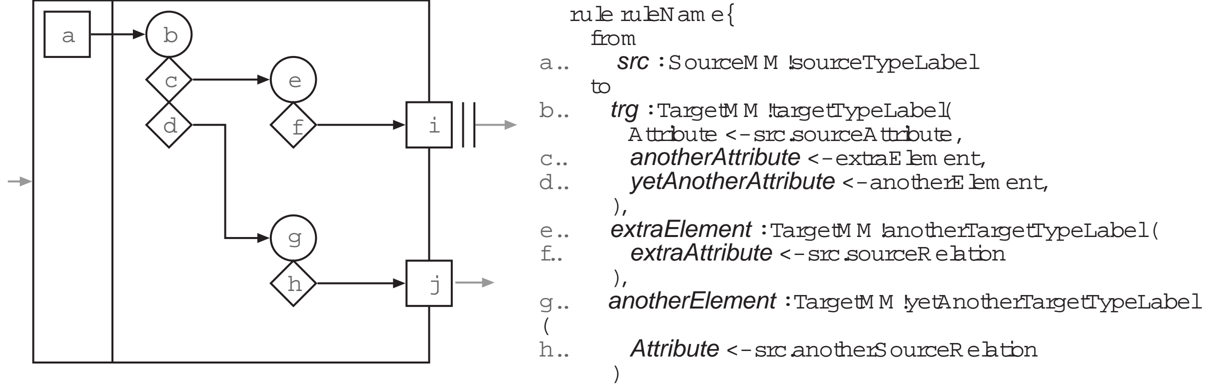

Figure 4.12 Composed diagram of a rule

notions work similarly for both left hand graphs and right hand graphs.

In ATL, as in other transformation languages, we get a notion of matching the left hand side elements of the rule to the source model, as to transform it to the representation on the right hand side. As source model and rule application conditions are describable as graphs, this notion of elements matching is formally described as a graph morphism, and represented in the diagrammatic analysis as shown in figure 4.9. This morphism is represented in a injective vertex map function.

For the definition of this morphism, let g be a graph, V_L' a vertex set and $h_V : V_L \to V_L'$ be a total injective vertex map. The edge map induced by h_V is $h_E((s,el,t)) = (h_V(s),el,h_V(t))$ for $(s,el,t) \in E_L$. The morphism $\hat{h}$ based on h_V is defined by

$$\hat{h}(g) = (h_V(V_L), h_E(E_L), l_L \circ h_V^{-1})$$

where $h_V(V_L) = \{h_V(v) | v \in V_L\}$ and $h_E(E_L) = \{h_E(e) | e \in E_L\}$.

If h is a bijection, then $\hat{h}$ is a isomorphism, and g and $\hat{h}(g)$ are isomorphic in symbols $\hat{h}(g) \cong g$.

Because there is no notion of matching elements with the right hand side, no morphism is defined, for the right hand side elements.

4.2.5 Rule Formalization

The matching of LHS elements defines the domain application of the transformation rules. These rules can be formalised as

$$Rule_k = (LHS_k, RHS_k, f_k)$$

where LHS_k defines the preconditions of application of the rule k, RHS_k defines the post-conditions of the rule k, and the function f_k relates the LHS application to the RHS application. Due to the complexity of sets in further defining the components of a rule, the following notation will be used: $X_{(w,z,k)}$ where X is an element (it can be a vertex, an edge or a labelling function), w denotes the element's position inside the rule (either left or right hand sides), z denotes the element's position on the symbolic propagation chain defined inside each rule and among them, and k denotes the exact rule where the element is defined. The symbol '_' denotes the available positions for all w, z and k.

4.2.5.1 Left Hand Side:

The LHS of a rule is defined as

$$LHS_k = (V_{(l,p,k)}, l_{(l,_,k)})$$

where $V_{(l,p,k)} \subseteq V_S$ and $l_{(l,_,k)} = l_L|V_{(l,_,k)}$ define the set of elements where the rule is applicable. The labelling function here is used to label all LHS vertices, including vertices from the LHS of the transformation rule which are referenced on the respective RHS of a rule. Furthermore, we partition the z group into three kinds of positions on the propagation chain: head vertex (p), middle vertex (m) and target vertex (t). Note also that head elements are heads of the propagation chain on each side of the rule, and by the Lhs_k definition, there can only be defined head vertices in LHS, resulting in an important restriction given from the ATL rules. Additionally, the matching of these head vertices can also be restricted by logical predicates expressed in

OCL rules, however the formalization of these rules are out of the scope of this paper - in [3] you can find an approach on the formalization of OCL rules in a theorem prover.

4.2.5.2 Right Hand Side:

In order to connect LHS head elements with RHS head elements, we define function f_k which relates a head element of LHS of rule k (denoted as $V_{(l,p,k)}$) with a head element of RHS of rule k (denoted as $V_{(r,p,k)}$).

$$f_k : V_{(l,p,k)} \rightarrow V_{(r,p,k)}$$

ATL allows the connection of the rules to each other by means of an association of a non terminal symbol defined on the RHS of the rule with some terminal symbol from the input model of another rule. The symbol references from the input model of another rule are usually expressed in ATL by using OCL constructs like 'allInstancesFrom' which are then composed with 'select' and 'any' constructs, to ensure the correct binding with the output of a determined rule.

To denote this notions, we have to define all the edges between vertices in V_S which were referred on the RHS of rule k, as $E_{(l,_,k)} \subseteq (s_l, el_l, t_l) \in E_L$ directed to elements in V_S, where the source vertex of the edges are defined as $s_l \in V_{(l,p,k)}$ and the target vertex as $t_l \in V_S$.

Since, in ATL, we can explicitly refer to a determinate input instance (instead of a relation which usually covers sets), we also define vertices which represent elements in V_S and were referred on the RHS of rule k, i.e $V_{(l,t,k)} \subseteq V_S$, where $V_{(l,p,k)} \cap V_{(l,t,k)} = \emptyset$.

4.2.6 Rule Composition

Finally the connection between rules is made clear with the definition of function g, which relates target elements of RHS of some rule k (denoted either as $V_{(l,t,k)}$ if it is an explicit vertex,

or t_l if it belongs to an edge) with head elements of LHS declared in rule j (denoted as $V_{(l,p,j)}$).

$$g_v : V_{(l,t,k)} \rightarrow V_{(l,p,j)}$$

$$g_e : t_l \rightarrow V_{(l,p,j)}$$

The propagation chain defined on the RHS of the rule is used to produce a graph on the RHS of the transformation rule set. After the execution of the rule, there was produced a set of vertices $V_{(r,_,k)} \subseteq V_R$, and a set of edges which connects them: $E_{r,_,k} \subseteq (s_r, el_r, t_r) \in E_R$. The sources of these edges are head and middle vertices $s_r = V_{(r,p,k)} \cup V_{(r,m,k)}$. The targets of these edges are middle vertices $V_{(r,m,k)}$, which represents explicit output terminals, plus the result of the interpretation of the implicit references to source model entities $V_{(l,t,k)}$, plus the result of the interpretation of the implicit references to source model relations t_l. The target of the edges are then defined as

$$t_r = V_{(r,m,k)} \cup f_j \circ g_v(V_{(l,t,k)}) \cup f_j \circ g_e(t_l)$$

.

The interpretation of target elements in rule k, is done by means of application of the result of function g in function f, given that function g matches target references with the appropriate rule, and function f matches source elements with target elements in the chosen rule.

Finally, the RHS of a rule is defined as

$$RHS_k = (V_{(r,_,k)}, E_{(r,_,k)}, l_{(r,_,k)}, E_{(l,_,k)}, V_{(l,t,k)})$$

where the RHS labelling function $l_{(r,,k)}$ is defined as $l_{(r,_,k)} = l_R | V_{(r,,k)}$.

In the figure 4.13, it is shown an example of an ATL rule, where $V_{(l,t,sys2ctxuse)}$ can also be seen as $V_{S_{preM}}$ bounded by some condition with t_l. In the example this is expressed by the composition of the OCL constructs 'allInstancesFrom' and 'any'.

```
module Layer02;
create coopnModel : COOPN refining hallModel : HALL, preM : COOPN;
                       f_sys2ctxuse: V(l,p,sys2ctxuse) -> V(r,p,sys2ctxuse)
rule  sys2cxtuse {
      from                                      V(l,p,sys2ctxuse)
            h : HALL!SystemComponent
                  (not h.componentSetInv.oclIsUndefined())
      to                                V(r,p,sys2ctxuse)
                                                                        V(l,t,sys2ctxuse)
            c : COOPN!ContextUse(
                  usedContext <- COOPN!COOPNContext.allInstancesFrom('preM')
            V(r,m,sys2ctxuse)     -> any( e |
                                        e.name =  'SystemComponent' + h.name )
                                                                     t_l
            )
}
```

Figure 4.13 An ATL rule sample decorated with the defined vertices $V_{(_,_,_)}$ and function $f_$ symbols, produced by the symbolic interpretation of the rule *sys2ctxuse*.

4.2.7 Algorithmic Application of the Formalized Rules

Now that we have the ATL elements defined, we can study the transformation process, based on ATL's default mode execution semantics description[19] and [9]. From this we take that the application of a transformation is divided in three successive phases: a module initialization phase, a matching phase of the source model elements, and a target model elements initialization phase.

- The first phase, initialization, Although not used in our particular study, ATL attributes that are defined in the context of the ATL module and subsequent dependencies, are initialized in this phase.
- In the next phase, matching of the source model elements, the source model patterns declared in the left hand side of the rules scope, are tested for a match subgraph on the source models. For this we use the morphism function $\hat{h}$, allocating resources for all isomorphisms detected. This allocation is composed by the target model instances declared for every matching rule, without any initialization.
- The final phase, target model elements initialization, applies the right and side of the rules to initialize the elements allocated in the previous step. Note that all element references

present in the rules are resolved in this phase, referring only to the finite set of allocated elements, ensuring termination of the ATL run in the presence of circular references.

4.2.8 Incremental Layer Specification

We define the notion of layer by expressing it in terms of a graph grammar. Each layer runs only once by executing all of its transformation rules, using the input from the previous layer and passes its output (and the remaining input) to be input of the next layer. Specifications are isolated at each layer. Each layer is a set of rules which deals each one with a disjoint set of input symbols and the execution of these rules occurs in a non-deterministic fashion. The transformation rule application algorithm can be viewed as one single layer in operation. When we have a sequence of different layers as shown in figure 4.14, the sets of rules in each layer are applied consecutively, where the output model of one layer (in addition to the original source model), is given as input to the next layer, in the manner presented in figure 4.15. In this sequence of layers, each one makes small (but visible) contribution to the overall transformation process.

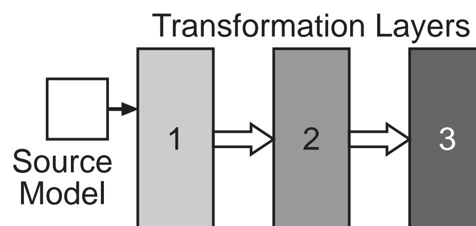

Figure 4.14 Layer chain

The formal definition of transformation layer is

$$Layer_q : (g_{s_q}, R_q) \rightarrow g_{t_q},$$

where g_{s_q} represents the input graph of the transformation layer, R_q is the set of rules that define the transformation in the layer, and g_{t_q} represents the output graph of the transformation layer.

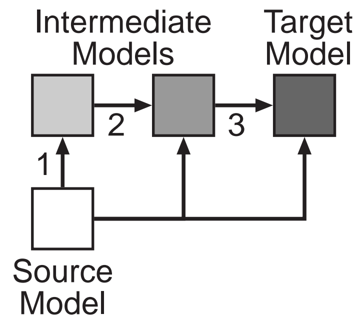

Figure 4.15 Incremental model approach

Furthermore, we show that for the first layer, no common elements can be found between the Source and Target models (this is given directly from the transformation rule definition).

$$Vs_0 \subseteq V_S$$

$$Vs_0 \cap Vt_0 = \emptyset$$

Also, the Source elements of the remaining layers will be composed by a mix of source elements and intermediate target elements, and the result of each transformation will only have intermediate target elements. A consequence of this is that if there are common elements in the Source and Target models of a layer, those elements must be the result from the previous layers.

$$Vs_q \cap Vt_q = Vt_{q-1}$$

The Source models of a layer can be composed of elements of both Source and Target elements of the full transformation process

$$Vs_q \subseteq V_S \cup V_T$$

However, the Target models of a layer can only be composed of elements of the Target elements of the full transformation.

$$Vt_q \subseteq V_T$$

The first noteworthy quality given by the layer abstraction is that it makes it possible to design the set of layers in a incremental way, hence enhancing the validation/navigation capabilities during the design process, where the model transformation designer is able to easily identify the layer (or sequence of layers) which is responsible for the consumption of a given input symbol (just by identifying the layer which is not propagating that symbol). The second noteworthy quality given by the layer abstraction, is that it can be used to effectively simplify the structure of logical predicates which forms the properties that the designer might want to satisfy while validating a set of model transformation rules. This simplification is made available by means of the definition of temporal partial order relations between transformation layers, where inside each layer there is no temporal relations of any kind between rules (i.e we cannot say which will be the first rule to be applied).

4.2.9 Semantic preservation

In our context of the transformation of computational-based semantic models, one important property that should be checked is the semantic preservation of the model transformation rules, i.e if there is a direct relation between entity A and B on the source language, provided that A is transformed to A' and B is transformed to B', then there should also be a relation (it might not be a direct one) between A' and B' on the target language.

In general, to demonstrate these kinds of properties, we often need to isolate the temporal expressions from what is to be guaranteed at each layer. Once the temporal partial order relation between transformations is discarded (as happens inside a transformation layer), we can focus our analysis in purely logical reasoning by evaluation of the paths found between the syntactic structures present on each transformation rule. The above property can thus be rewritten to 'if there is a path between two elements in the input left hand graph, then there must also be a path between their respective elements on the output right hand graph'.

A direct path between two nodes is defined as the existence of an arch which directly connect

them. This is written as: for a pair of nodes (A,B) there is a direct path $path(A,B)$ if

$$inc(A) \cap inc(B) \neq \emptyset$$

For a broader definition of path, between A and B we take that if there is a direct path between A and C and there is a direct path between C and B, then there exists is a path between A and B. This is written as

$$path(A,C) \wedge path(C,B) \Rightarrow path(A,B)$$

Note that this notion decomposes any path as a composition of direct paths. This notion works similarly for both LHS graphs and RHS graphs.
To assure semantic preservation of the transformation process, we analyse both the paths on source and target models, and how these paths connect each other inside and among the defined transformation rules. Therefore, we must assure that if there exists a $path(A,B)$, where $A \wedge B \in V_S$, then there must exist at least a rule k with $A = V_{(l,p,k)}$ and a rule j with $B = V_{(l,p,j)}$, and a association function g which relates either t_l or $V_{(l,t,k)}$ with $V_{(l,p,j)}$. This ensures that for every possible source model of the transformation rule, if there exists a directed path between its vertices, then there also must be a directed path between the correspondent vertices on the generated target model.

4.2.10 Confluence

It is important to remember that the execution of the transformation rules is done in a non-deterministic fashion inside a layer. Therefore, in order to guarantee confluence i.e uniqueness on the results of the model transformation, we have to impose new restrictions to the application of the transformation rules. If K is the set of all rules present in each layer, then:

$$\forall i \in K, V_{(l,p,i)} \cap \{ \bigcup_{j \in K \setminus \{i\}} V_{(l,p,j)} \} = \emptyset$$

4.2.11 Completeness

To achieve completeness in the overall model transformation, i.e if all entities of the source language must be translated to entities on the target language [1], then we must assure that for all $v \in V_S$, there must be at least a layer q with rule k which defines v as $V_{(l,p,k)}$:

$$\forall_{v \in V_S}, \exists_{Rule_k \in Layer_q} : v \in V_{(l,p,k)} \wedge Lhs_k(V_{(l,p,k)}, l_{(l,_,k)})$$

[1]Note that in some cases in model transformations, we might want to ignore some useless entities or their relationships.

5 . (H)ALL Language Engineering

AS already mentioned in chapter 2, in a first stage of the design process of (H)ALL, it was established in [2] its requirements and terms, of the complex control system domain, that compose (H)ALL's application domain. With this, a first proposal for the language's specification was done and completed, with meta-models to define its syntax and syntax-to-syntax transformation proposals in QVT, into the CO-OPN language, in order to define its semantics .

We will now focus in this chapter on the subsequent steps needed to implement the (H)ALL language. To do this, we need to look into the specification of (H)ALL and adapt it, to the capabilities of the chosen language workbench technology, while trying to be faithful with the original design.

These adaptations are reflected in both the syntax and semantic of the language. The first through the specification of the language in a meta-modeling tool and its use restrictions. The second, through the definition of transformation rules in terms of the target framework, CO-OPN builder.

5.1 Syntax

As in any Domain Specific Visual Language, (H)ALL's syntax is defined with a meta-model. This meta-model is the key point in the construction of the language's editor. The meta-model is in our case, as in most cases, defined by a MOF compliant diagram. This is done by specifying with UML like classes the corresponding elements available in the language, and with associations, how they can be combined. However irrelevant for this thesis, as already explained in chapter 4, further restrictions can be applied to the language through the use of OCL, by specifying the cases where the meta-model would allow for the creation of valid models, in the context of the language application.

5.1.1 Meta-Modeling Process

What we call Meta-modeling process will start by taking the language specification made in [2] and adapt it to the MOF compliant ecore meta-model format. As the (H)ALL's language syntax was specified by means of several meta-models, which focus in each particular characteristic of the language, we need to compose all these meta-models in one unique meta-model that specifies the language as a whole. As the meta-models were made in complement to each other, they allow for a direct combination of classes through substitution of common elements(grayed elements in the shown figures) or by aggregation.

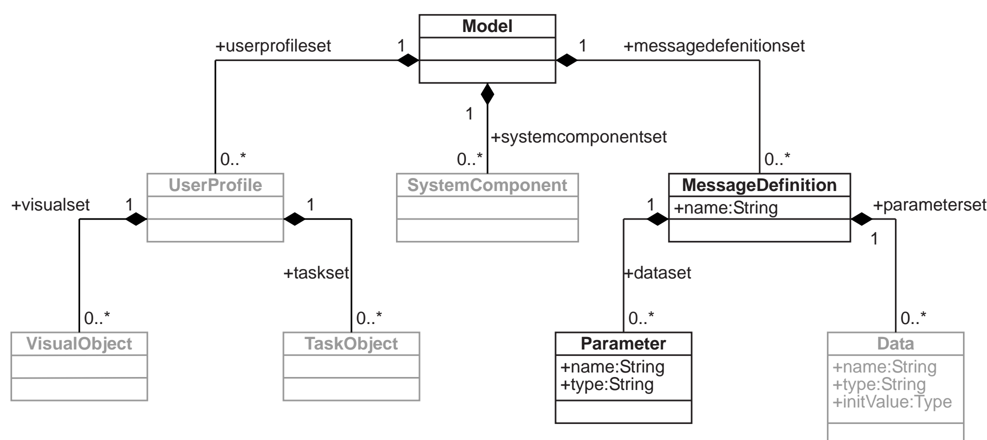

Figure 5.1 Root meta-model

The substitution of common elements is done when a class, that represents the same aspect of the language, is present in more than one of the initial meta-models. An example of this is the *UserProfile* class, present in the Root meta-model(figure 5.1) and the Component inheritance meta-model(figure 5.2). In this case, we fuse the multiple instances into one single instance, with the attributes and relations present in every one of them, replicated in the single instance. We then replace the instances, with the substitution class, forcing the union of the meta-models involved. Other Classes where this occurs are, *Component*, *SystemComponent*, *TaskObject*, *VisualObject* and *Data*.

The aggregation of meta-models, occurs when there is not a direct correspondence between

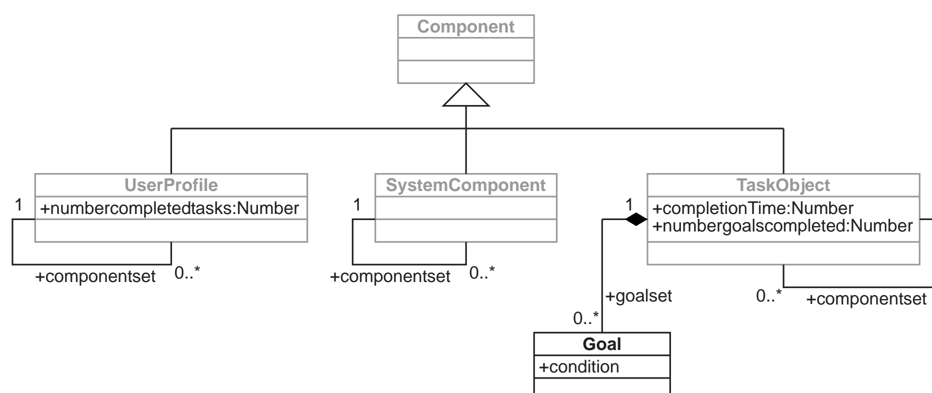

Figure 5.2 Component inheritance meta-model

classes present in the meta-models, but they still relate. This is the case of the *PreConditionExpression*, that has a meta-model representation(figure 5.3), and is referenced in Component as an attribute(figure 5.4). In this case we remove the referencing attribute, and replace it with a direct aggregation relation. Because of the sub-editors expressiveness, and to isolate the Expressions, we place a containment class between the referencing class and the referenced meta-model as the class *Expression* in figure 5.4. Other elements where this occurs are, the *Trigger*, PosConditionExpression, *ActionExpression*, *PreConditionMessageExpression*, *PosCoditionMessageExpression* and *ActionMessageExpression*.

s

The next step is then to chose a class to serve as a representative of the language that contains all elements during the model editor execution. The best candidate for this task was the *Model* Class(figure 5.1) Because GMF allows the evocation of new editor definitions within a model, we divide the modeling language for readability purposes. Instead of dividing it into several different meta-models, we leave it as one single meta-model, with different editor definitions for each section or subgroup. For this we must alter the meta-model so that it can have a representative of the subgroup of the model, for all subgroups, choosing to add a new one when no class originally present in the language definition presented itself as suitable for this task. This is the case of the Expressions meta-models, such as the *PreConditionExpression*

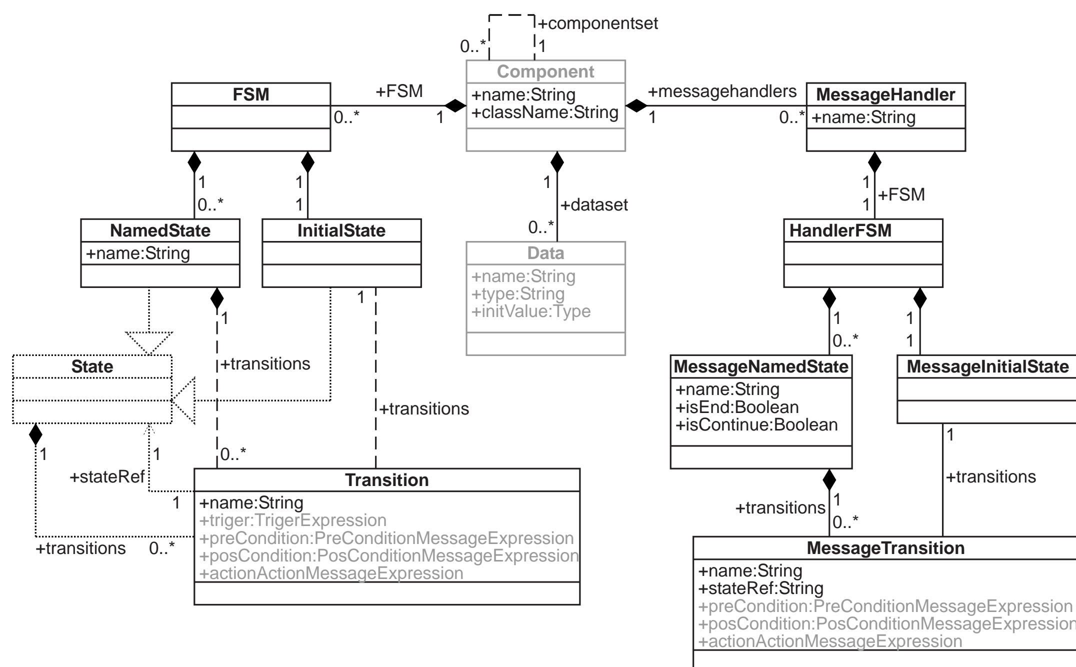

Figure 5.3 Component meta-model

meta-model(figure 5.4), where the *PreConditionExpression* class was not part of the original specification, being added for the implementation reasons here presented.

As a final step, we optimized the meta-model, removing redundant class entries, such as the *HandlerFSM* in figure 5.3, or when a group of classes shared characteristics, driving them from inheritance, rather than specifying them for every class, as in the *State* class in figure 5.3.

The application of these adaptations leads to the language specification defined by the complete meta-model, that because of its visual complexity is shown in figure/appendix A.

5.1.2 Editor Generation

With the language meta-model adapted for the editor generation we look at the implications of using multiple editors and their implementation . We start by reviewing the relations between these editors, as viewed in figure 5.5. This step is important because, for every editor, we have to specify all referenced editors, including in some cases the referencing editor itself,

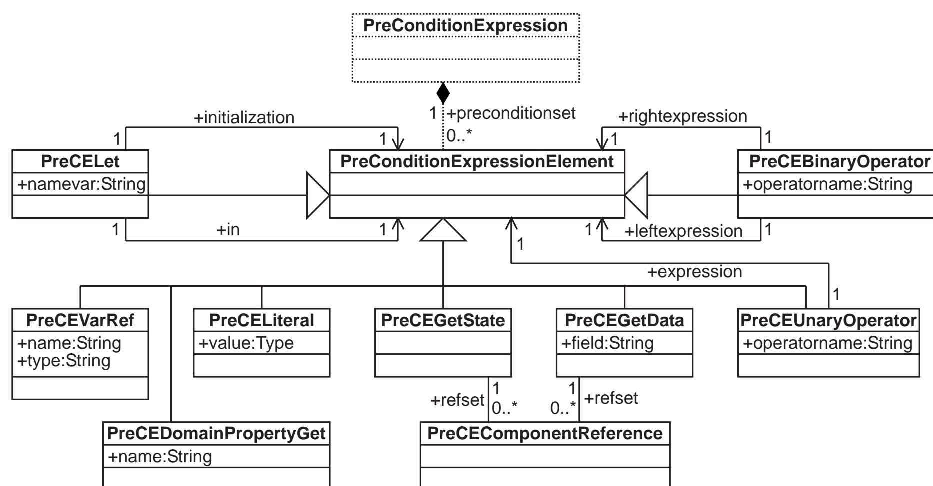

Figure 5.4 PreConditionExpression meta-model

which can make it hard to process and maintain. For this reason we maintained annotations on the referencing process as presented on Appendix C. After we established the implications every editor has on the other editors, we can proceed by generating the editors from bottom up, updating the already created editors, or by generating them top-down, but working in the inverse order of the models. At this point in development we chose the first approach.

For the creation of each editor we follow the guidelines present in figure 5.6. The next subsections describes the particularities of those steps applied to the building process of the (H)ALL editor.

5.1.2.1 Creating the Domain Generator model

The first step is to derive the domain generator model. We then make a copy of this model for every editor that we will create. For every model copied in this way,we need to correct a given set of parameters with their editor specific values.

At the root of the domain generator model we change the *ModelID* and *DomainGenModel* parameters to the identification of the specific editor and in the *PackageNamePrefix* parameter, we add *.diagram* to that identification. We then proceed in the model hierarchy to GenPlugin,

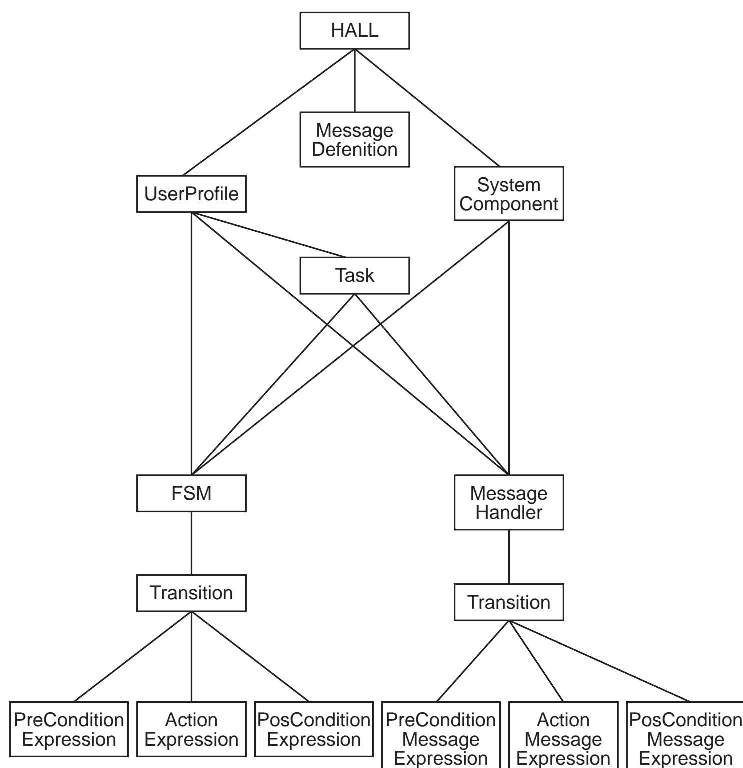

Figure 5.5 Inter editor dependencies

where we change the *ID* parameter to the same present in *PackageNamePrefix*, making the *Name* and *ActivatorClassName* parameters change automatically.

The last step in the domain generator model is to make the elements prone to relate to other editors. For this we go to the *GenDiagramModelEditPart* section of the hierarchy and in the elements that provide the contact with the next set of editors we go to the *OpenDiagramBehaviour* component of their hierarchy and set the attributes *DiagramKind*, *EditId* and *EditPlocyClassName* to the identification of the invoked editor, the composed identification of the diagram editor in the following manner identification.*diagram.part.*identification*DiagramEditorID* and the identification followed by *DiagramEditPolicy* respectively.

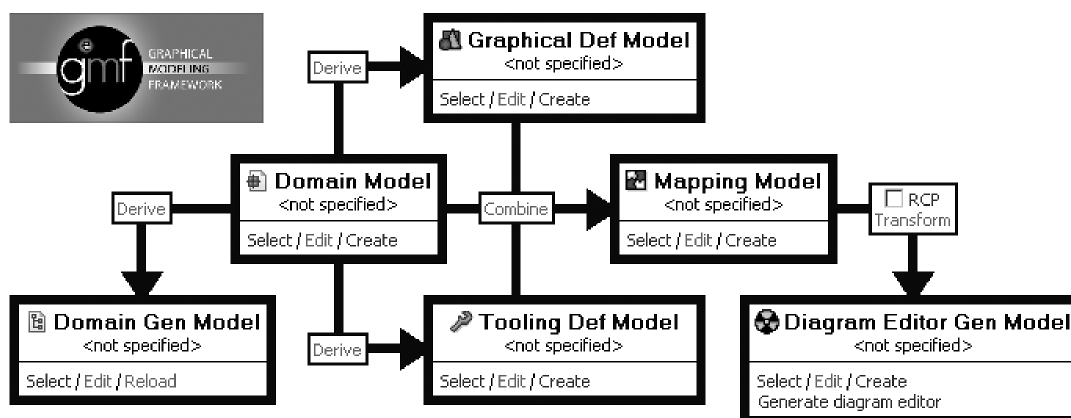

Figure 5.6 GMF Dashboard

5.1.2.2 Creating the Graphical Definition Model

For this step we drive the graphical definition model, selecting only the elements present in a given editor. We then need to correct any visual representation that is different from the default, and add a label component for all elements that have no explicit identification in the meta-model.

5.1.2.3 Creating the Tooling Definition Model

This is the simplest phase of the process as we only need, for every editor, to select the corresponding elements specified in the visual mapping model, organize them in a proper manner and give them a expressive icon, by replacing the original image with the new one.

5.1.2.4 Creating the Mapping Model

We then combine the derived models and the meta-model, into a mapping model. This is the most complex part of the process, as the correspondence between all of these models is made here. Because of this we need to verify that for every element , the corresponding tool and visual specification is correctly associated. To do this, we need to check that the names refer to

same element. In all elements we added a label for visual purposes needs to be added here to, but instead of a *feature label mapping*, we add it as *design label mapping*. Finally, the mapping to relate the editors has to be done here as well. For this last step we include the mapping for every targeted editor, and then navigate in the mapping hierarchy to the *TopNodeReference* of the editor invocable notes to set the *NodeMapping*'s *RelatedDiagrams* attributes to the canvas mapping of the targeted editor. In this last step we may need to check in the XML file if the correct canvas mapping is being used.

5.1.2.5 Creating the Editor Generator

In this step we only need to verify if any errors where raised, and if so, correct them. In most cases, these errors are originated in the mapping model. When no errors are raised we can generate the code for the editor creation. The created editor will be similar to that presented in figure 5.7.

5.1.3 Language Reprocessing

From this process we have created a working editor and a set of feedbacks that allow us to return to the language specification, to optimize and improve its implementation. One of such cases is the inclusion of several sub editors and the consequent adaptation of the meta-model to support this. Additionally both user interaction and framework evolution, contribute to the improvement process of (H)ALL.

5.2 Semantics

In Domain Specific Modeling, semantics assumes a dual role, in a top-down perspective, it represents terms, meanings and what is expected from a DSL model under the perspective of the

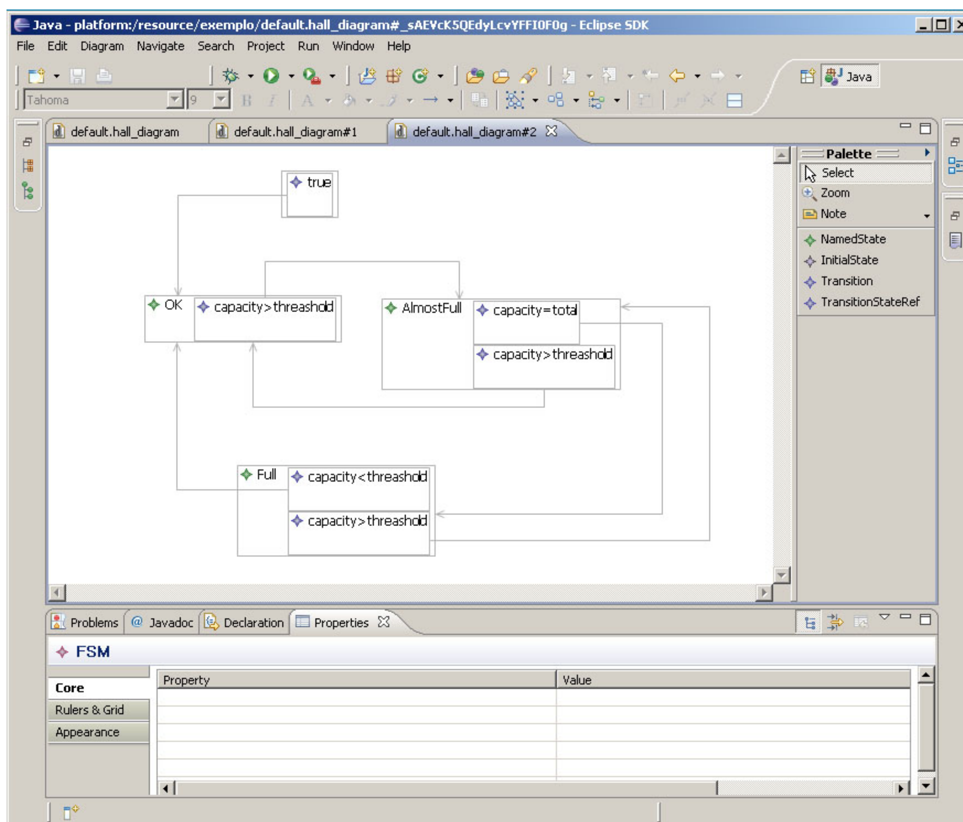

Figure 5.7 (H)ALL editor interface

modeler, in a bottom-up perspective it carries all the meaning of semantics in a computational and language development point of view. In developing a language that has its objectives in the abstraction of systems, it is hard to accurately choose between these two perspectives.

To deal with this semantic definition duality, we analyzed the source and target patterns for their semantic meanings, to ascertain the correct perspective to take. From this, we concluded the existence in the source and target models, of a mixture of patterns that owed their semantics to the source platform, patterns that owed their's to the target platform, and some patterns could be seen as having their semantics given by both the source or target platform. Finally, we separated the model patterns into three groups:

- **Source group**, where the patterns owed their semantics to the source models, such as the

relation between hierarchic elements of the specified interface. These patterns are defined by the domain expert describing the control system.

- **Target**, where their semantics was defined by the target model in the transformation process, such as the process of message handling between elements. These patterns are defined by the tool developer, and are set automatically.

- **Neutral**, where all elements whose semantics could be seen as defined by the source or target model, and the transformation process brought no alteration to it, such as the ADT types used in both source and target models.

In the first group of patterns, we needed to ensure that the semantics already present in these patterns will be maintained. To ensure this, a verification of these patterns was made, and is presented in section 4.2.

The second group of patterns, because it is being defined in the transformation process, has no point of comparison, to ascertain their correctness, unlike those present in the first group. To deal with these a judgment call is made by justifying the patterns being created and the semantics that is being added to them.

As for the third group we only need to ensure the use of common ADT type references during the transformation process. This occurs because, maintaining the type, the content and operational semantics will be maintained intact.

Taking these groups of patterns into account, and to further enrich the specification model's semantics, we build the transformation rules to deliver the models into the target framework.

5.2.1 Transformation Rules

We proceeded, after the phase were we generated the (H)ALL editor was completed, with the creation of the transformation rules, but because of their huge number and complexity are shown in appendix D. These rules are composed by a *header*, a *from* and a *to* sections, as shown in

figure 5.8. The *header* section defines the unique identification of a rule. The *from* section describes the pattern present in the source models, that will trigger the application of the rule. The *to* section of the rule specifies the target model elements created with the application of the rule, and any attributes and relations, that those elements might have.

```
rule userprofiletocontext
  from
    h : HALL!UserProfile
  to
    c : COOPN!COOPNContext(
      name <- 'UserProfile' + h.name,
      ownedBody <- b,
      ownedInherits <- d
    ),
    b : COOPN!Body (
    ),
    d : COOPN!Inherit (
      inheritedContext <-
        COOPN!COOPNContext.allInstancesFrom('preCoopnModel')->
        any(e | e.name = 'GenericUserProfile')
    )
```

Figure 5.8 ATL rule example

We begin defining the transformation of (H)ALL to CO-OPN elements, by specifying the construction a global CO-OPN Package that contains the full model, and specifying that all hierarchic elements will be translated as contexts contained by the package, and that any (H)ALL element contained by the hierarchic elements, and subsequent containments, would be translated as a CO-OPN Object to be contained by the corresponding CO-OPN Package or Object. Data types wold be translated into their ADT correspondent.

To better control the transformation process, the created rules are grouped by context into different files. These files are designate as layers, and are applied in sequence to the source model and any intermediate stage of the process. In this manner, we can apply an incremental approach to the construction of the target model, where each new step adds to the solution without losing the already created elements of solution. In figure 5.9 we can observe the rule that maintains the incremental construction of the models, when proceeding in ATL's refinement mode, by copying the root element. This is made possible because in refinement mode ATL copies all implicit model elements that relate to the elements referenced in the rule, and this particular rule forces the accessibility of the complete model by referencing the root element, that relates to the full model.

```
rule packageRefactor
  from
    s : COOPN!COOPNPackage
  to
    t : COOPN!COOPNPackage(
      name <- s.name ,
      ownedModules <- Sets.ownedModules
    )
```

Figure 5.9 ATL propagation rule

This is necessary because we cannot build the target model in a single transformation step. For instance, to start building the models, and because there is no direct type translation of the hierarchic elements of (H)ALL to CO-OPN, we first create a layer that builds representative elements of these types, so that during the remaining target model construction, they already exist and can be referenced.

Another case where consecutive layer application is necessary, is when a source pattern generates multiple independent target patterns. This occurs because each target pattern specifies

an isolated characteristic of the source pattern, but can only be run once per layer. In figure 5.10 we have an example of a rule that wold conflict with the rule presented in 5.8, because they are driven from the same *from* pattern, and references the elements produced in the first rule, and so it is executed in the next layer.

```
rule userprofiletocontextuse
    from
        h : HALL!UserProfile
    to
        c : COOPN!ContextUse(
            usedContext <-
                COOPN!COOPNContext.allInstancesFrom('preCoopnModel')->
                any( e | e.name = 'UserProfile' + h.name )
        )
```

Figure 5.10 example of an ATL rule that requires the use of layers

Most of these applications occur when specifying lower level details from the higher level model patterns, and when specifying semantics not yet present in the models. This is the case of the message handling between elements, where at a higher level was abstracted how this wold occur, but at a lower level needed to be made explicit. At this point we defined how the message handling between elements would occur. From several ways to handle the messages, we choose the creation of a router object within every element's context. This router checks the message's signature for particular handling patterns corresponding to the element's task. If the pattern is present in the message signature, it is handled by the present element, and computation proceeds accordingly, if not the message is propagated to all elements present at the next level of propagation. This way the router takes the same procedure with messages going up in the hierarchy or going down, for this is handled outside the router objects. From the outside of the contexts, propagation axioms are built according to the hierarchic structure

defined in the source model in both propagation directions, but connected at the same gates, this way the axioms can select themselves the propagation to be taken by the message, based on the message's signature.

5.2.2 Rule organization

Due to the large number of rules, the layer files, where organized according to the context of the rules included in each of the layers and the dependencies between them in a incremental manner. And so the layers have the following structure:

- Layer **Declarative 01** holds the base elements for the target model construction, with the definition of the model context, domain types and ADT
- Layers **Declarative 02** and **03** defines the transformation of the hierarchic types
- Layers **FSM 01** through **03**, build the Finite State Machines of the transformed hierarchic elements, for definition of behavior.
- Layers **mfsm 00** through **03**, build the Finite State Machines of the transformed hierarchic elements, for definition of message handling behavior.
- Layers **Data 01** through **05**, set the source model data in terms of target model ADT's.
- Layers **Router 01** through **05** define the message handling propagation for each element created.
- Layers **Router 06** and **07** define the message handling propagation between elements.

6 . Validation

Upon the creation of the editor tool, and establishing the transformation process rules, we produced a set of tests to verify the target models automatically generated by our tool. These will add to the verification of the syntax and semantics evaluated the chapter 4.1 and 4.2.

With the implemented (H)ALL editor, we start by creating models that specify our system and check if it is capable of expressing them. Afterwards, we use these same created models to verify the transformation process, by observing if the target models correspond to what would be created if we were producing them directly in the target framework.

To proceed with this testing phase, we establish two sets of tests. The first one, composed of simple models, has a set of minimalistic patterns, to observe if all elements of the meta-model were being taken into account, and to observe the behavior of these patterns in the presence of each other. These tests deal with the patterns themselves in controlled conditions, not necessarily realistic, and not with the large scale description of the systems. The second set of tests, corresponds to a real life case study, of medium complexity where an existing system is specified and we check if the corresponding result copes with the existing solution, or if it is completely different. In this last case we then observe if these differences arise from frailties in the existing solution or if it is a fault in our design expressiveness or the transformation processes.

6.1 First Set of tests

These tests are composed of combinatory patterns, where every simple pattern is specified, and the combinations between these simpler patterns is tested, resulting in the combination of all visible patterns presented in section E.1 of appendix E.

With these tests, we aim to achieve a verification of expressiveness on the editor, by proving

that we can reproduce any design-time pattern with it, and to verify that the result of the transformation process over these patterns, achieves the expected target patterns. By combining the simpler patterns with each other we can observe the influence they have and if they interfere with the good development of the overall system.

Because the application of the transformation layers is incremental, the inclusion of new patterns was also incremental. In doing this we tested the first layers, and then incremented the tests with new patterns, designed to test new layers. When running the new tests, we checked that the already tested layers maintained the same results, and then verified the results on the target layers(regression tests). We proceeded in this manner until, all minor patterns where used in testing the full meta-model, and all layers had been tested.

These tests where taken during the development of our work and their results where largely taken into account in correcting and improving the obtained solution.

Upon this we achieved a point where we were satisfied with the test set results, leading to the nest step of using a real life case study.

6.2 Case Study

As a case study we observed and recreated the ATLAS experiment GUI. The objective of this recreation is to observe if we can specify a real system, and compare our results with the existing implementation. With this comparison, we expect to verify the expressiveness of the current implementation of (H)ALL language, and check if there are any elements of the development process that allowed a clearer solution.

6.2.1 ATLAS

The ATLAS experiment is a High Energy Physics(HEP) experiment being held at CERN. The elevated number of components and hierarchic structure, make the development of this experiments GUI a perfect candidate to test the application of our work.

The particular focus of our case study is the Online Software of ATLAS, that controls the Data Acquisition System(DAQ), and its general purpose Graphical User Interface(GUI). In figure 6.1 we can see a layout of this GUI, composed the Main Commands & Display panel on the left, were we can observe the most important elements and parameters of the system, on the right we have a set of tabbed panels that allow the interaction with the different parts of the DAQ system, and in the lower section the MRS message window, where all type of messages are displayed. Our validation purpose is to replicate the GUI, by specifying the DAQ and how the GUI handles the its events.

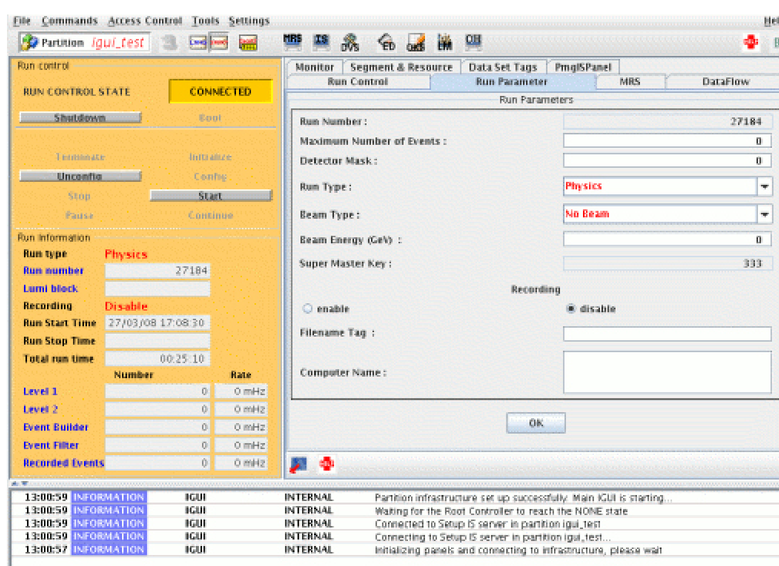

Figure 6.1 general purpose User Interface for the Online Software of ATLAS from [20]

6.2.2 Procedure

An external member to this particular stage of development of our thesis, used the created tools to specify the GUI used in the ATLAS experiment. This gave us an independent look over the created tools, and allowed to verify the tool's usability. From this we have the model presented in appendix E.3, that we used for testing.

In it we can observe the specification of a GUI, for a particular user, as shown in figure 6.2, with a specification of a *MainPanel* that aggregates all GUI elements, a *TreePanel* corresponding to the Main Commands & Display panel of our reference GUI, a *TabbedPanel* corresponding to the tabbed panel for the interaction with the different parts of the DAQ system, composed by some structural elements of that panel to display information and enable interaction with the system.

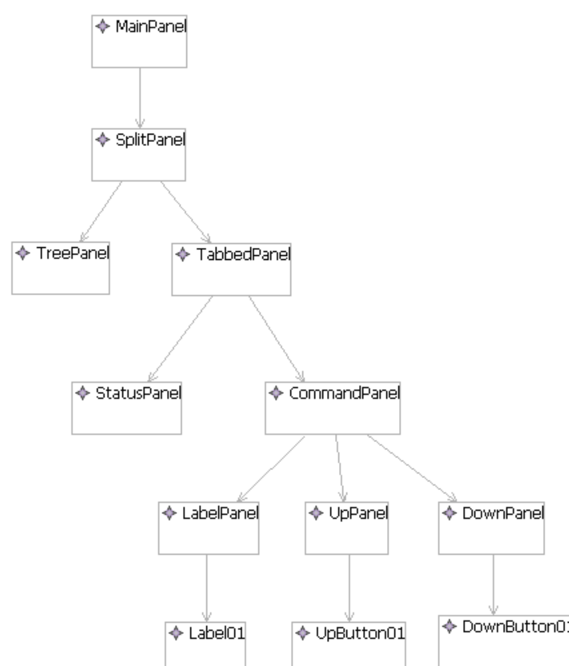

Figure 6.2 Specification of the Visual Components of a user's GUI

In figure 6.3 we have the hierarquic specification of the DAQ structure and components, so that they can be represented in the *TreePanel* for status information display and interaction with

the system.

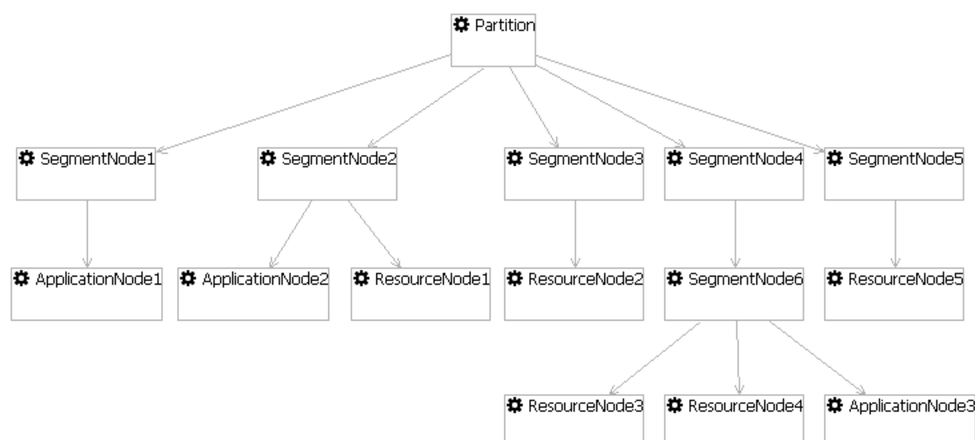

Figure 6.3 DAQ system and structure specification

The behavior of these components in the *TreePanel* is defined by a Finite State Machine, that specifies all states of a component and how they relate. In figure 6.4 we can observe the specification of this FSM.

After the specification of the system'a GUI, the model is carried to CO-OPN through the transformation process. Where for instance, the XMI description of the tree panel present in figure 6.5 is turned into the XMI/CO-OPN context present in figure 6.6 at layer Declarative02. This context is created with an empty body and a inheritance reference to the respective (H)ALL type representative context.

At layer Data01 the *TreePanel* data representation presented in figure 6.7 is added to the target model. This is done with a class module, with a generic variable set to allow it to reference itself, and a definition of its interface.

In layer Data02 the data class object is added to the context's body composition as highlighted in figure 6.8.

In the final stages of transformation, a router is created to handle the message exchange between the specified elements. This router creation is initiated in layer Router01, and is presented in figure 6.9, with the creation of a place to hold the routers running status, a set of

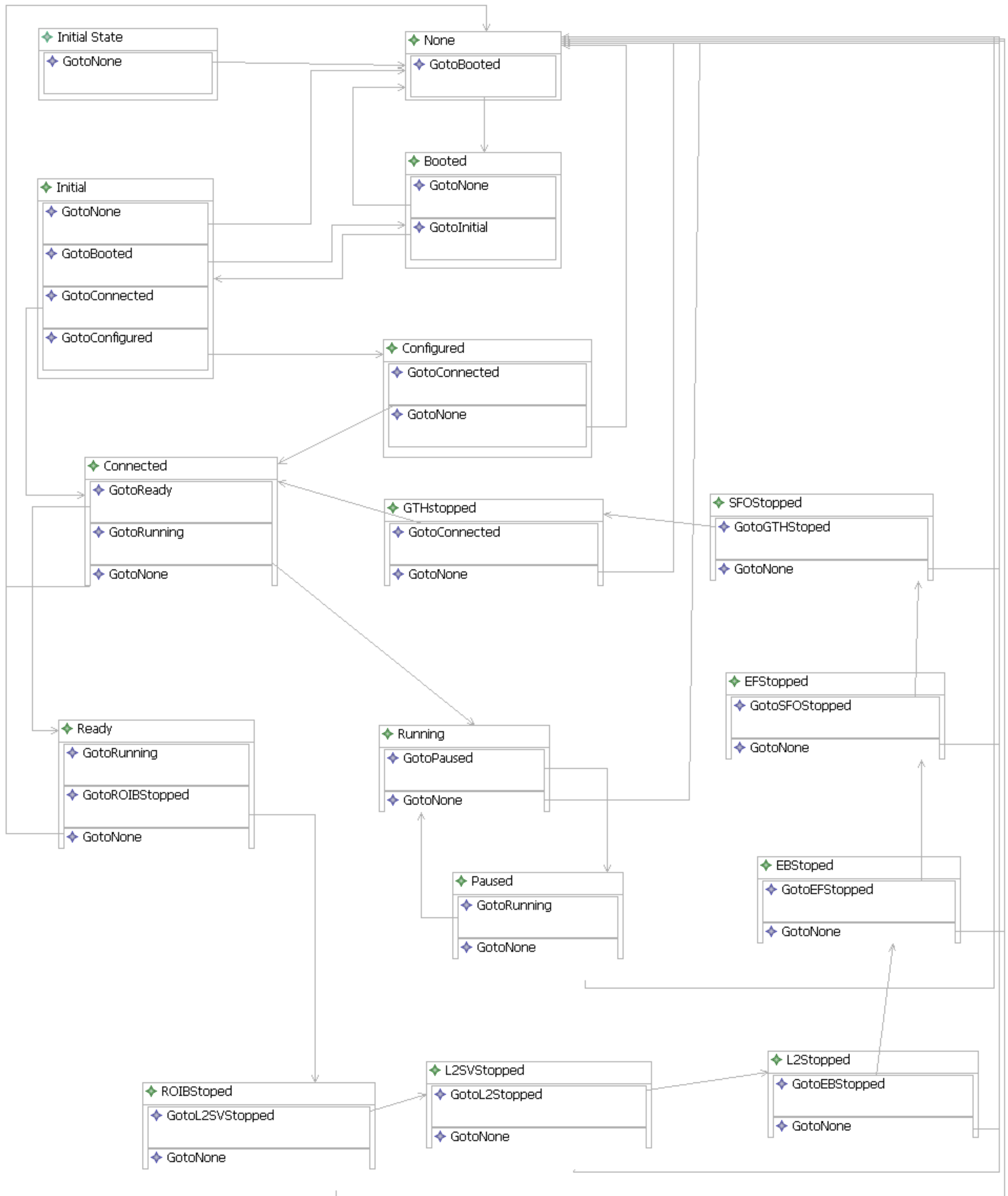

Figure 6.4 FSM for component control

```
<visualObject name="TreePanel"
        componentSetInv="//@userProfile.0/@visualObject.1"/>
```

Figure 6.5 TreePanel XMI (H)ALL representation

```
<ownedModules xsi:type="COOPNMetaModel.ContextModule:COOPNContext"
        name="VisualObjectTreePanel">
    <ownedBody/>
    <ownedInherits inheritedContext="//@ownedModules.20"/>
</ownedModules>
```

Figure 6.6 TreePanel target context representation at layer Declarative02

variables to allow the checking of the messages signature, the specification of the router's status on initialization of the system, and the interface for the router's interaction with other elements of the interface.

After the processing of all of these stages, the (H)ALL model specification was successfully carried into the CO-OPN framework, as we intended to demonstrate.

6.2.3 Results

With the application of the case study, the overall usability of the developed tools was proven positive, and gave us confidence on the developed work, and that it is in the right direction. Some issues where found at the tools framework level, such as memory management faults, and limitations when used as a stand alone plug-in, but none of these depended on the specification of the tool itself, as all of the editor's code is automatically generated, and so it remains to wait for the correction of these issues with the evolution of the eclipse framework.

```xml
<ownedModules xsi:type="COOPNMetaModel.ClassModule:COOPNClass"
            name="VisualObjectTreePanelData">
    <ownedBody>
        <ownedVariables name="this" variableType="//@ownedModules.74/
            @ownedInterface/@ownedInterfaceClassTypes.0"/>
    </ownedBody>
    <ownedInterface>
        <ownedInterfaceClassTypes
            name="VisualObjectTreePanelDataType" order="1"/>
    </ownedInterface>
</ownedModules>
```

Figure 6.7 TreePanel target data representation at layer Data01

```xml
<ownedModules xsi:type="COOPNMetaModel.ContextModule:COOPNContext"
            name="VisualObjectTreePanel"
            contextUse="//@ownedModules.33/
                @ownedBody/@ownedContextUses.18">
    <ownedBody>
        <ownedObjects name="VisualObjectTreePanelDataObject"
            objectType="//@ownedModules.74/
                @ownedInterface/@ownedInterfaceClassTypes.0"/>
    </ownedBody>
    <ownedInherits inheritedContext="//@ownedModules.29"/>
</ownedModules>
```

Figure 6.8 TreePanel target context representation update at layer Data02

```
<ownedModules xsi:type="COOPNMetaModel.ClassModule:COOPNClass"
            name="VisualObjectTreePanelRouter">
    <ownedBody>
        <ownedPlaces name="VisualObjectTreePanelRouterEnabled">
            <ownedPlaceTypeElements typeElementType="//@ownedModules.68/
                @ownedInterface/@ownedInterfaceSorts.0" order="1"/>
        </ownedPlaces>
        <ownedVariables name="mdirection" variableType="//@ownedModules.10/
            @ownedInterface/@ownedInterfaceSorts.0"/>
        <ownedVariables name="this" variableType="//@ownedModules.109/
            @ownedInterface/@ownedInterfaceClassTypes.0"/>
        <ownedVariables name="mname" variableType="//@ownedModules.77/
            @ownedInterface/@ownedInterfaceSorts.0"/>
        <ownedVariables name="mparam" variableType="//@ownedModules.83/
            @ownedInterface/@ownedInterfaceSorts.0"/>
        <ownedInitial>
            <ownedInitialTerms
                xsi:type="COOPNMetaModel.ContextModule:Term"
                expression="VisualObjectTreePanelRouterEnabled @"/>
        </ownedInitial>
    </ownedBody>
    <ownedInterface>
        <ownedInterfaceGates name="fireTreePanelRouterPost"/>
        <ownedInterfaceMethods name="fireTreePanelRouterPre"/>
        <ownedInterfaceClassTypes
            name="VisualObjectTreePanelRouterType" order="1"/>
    </ownedInterface>
</ownedModules>
```

Figure 6.9 TreePanel target router construction at layer Router01

7 . Conclusion and Future Work

With the completion of this thesis we proceed with the analysis of the development process, the setbacks felt with that process, obtained results and we propose future directions for the evolvement of this work.

7.1 Conclusions

In general, as seen in the previous section, this approach to complex control systems, has shown to work and to be able to produce correct results.We were able to effectively reduce the gap between the design stages of the solution created on the BATIC3S project and their implementation stages.

At a first stage of our work, we collected information on the expressiveness and use of visual editors in specifying User Interfaces to Control Complex Systems, with a variety of different components and with different characteristics. As we needed to implement a formally designed language in these tools, to actively specify the User Interfaces, we had to re-design certain sections of (H)ALL in order to adapt it to the technological limitations. During this process, the rapid development cycle of the methods used, allowed for a coherent construction of the tools, where these were updated without the use of patchwork or hard to maintain development.

From building this tool we iterated on the original expression paradigms of specification as to improve future user interaction. We also verified the syntactic guaranties offered by this type of tools, where the model creation process verifies the models for correctness as they are being created, avoiding the introduction of errors in the development cycle. In the study of these syntactic guaranties, we also observed that, once guarantied to the initial development stages, they can be maintained throughout the remaining development process.

At a second stage, with the verification of the transformation process, we were able to produce a more secure set of target models, in relation to their design specification. Although

the transformation methods used where limited, to the declarative paradigm, as it limits the transformation rules application and model navigation,we were able to overcame this limitation to maintain a significant level of expressiveness, by using sequential sets of rules we called layers, and with this maintain a more transparent verification of the transformations. Given the good results obtained with this method and its verification, it became one more element of this work that we foresee that can easily be carried into other domains of development.

By implementing the model transformation rules from (H)ALL to CO-OPN with our proposed transformation method, we were able to produce, as shown in the previous section, a complete development cycle in the specification of User Interface of control systems for complex large scale systems, and deliver it to the subsequent stages of development. Not only in effective specification and transition to the next stages, but also ensuring the quality of the delivered models. With the further development of the target platform and auxiliary components in the BATIC3S project, the production of complex control systems User Interfaces, will become a design compliant process, allowing the definition of the control system along with the controlled system, through the use of rapid prototyping, improving the development cycle of not only the control system User Interface, but also the control system itself.

7.2 Future Work

From this point on we will work on producing an automated verification of the soundness of the transformation rules through the use of theorem provers, and produce a new editor with improved usability and stronger expressiveness. Furthermore, we believe that the transformation methodology proposed in this thesis can be generalized outside the complex control systems domain to further increase their strength and their use.

A . (H)ALL Metamodel Diagrams

Figure A.1 (H)ALL Meta-Model

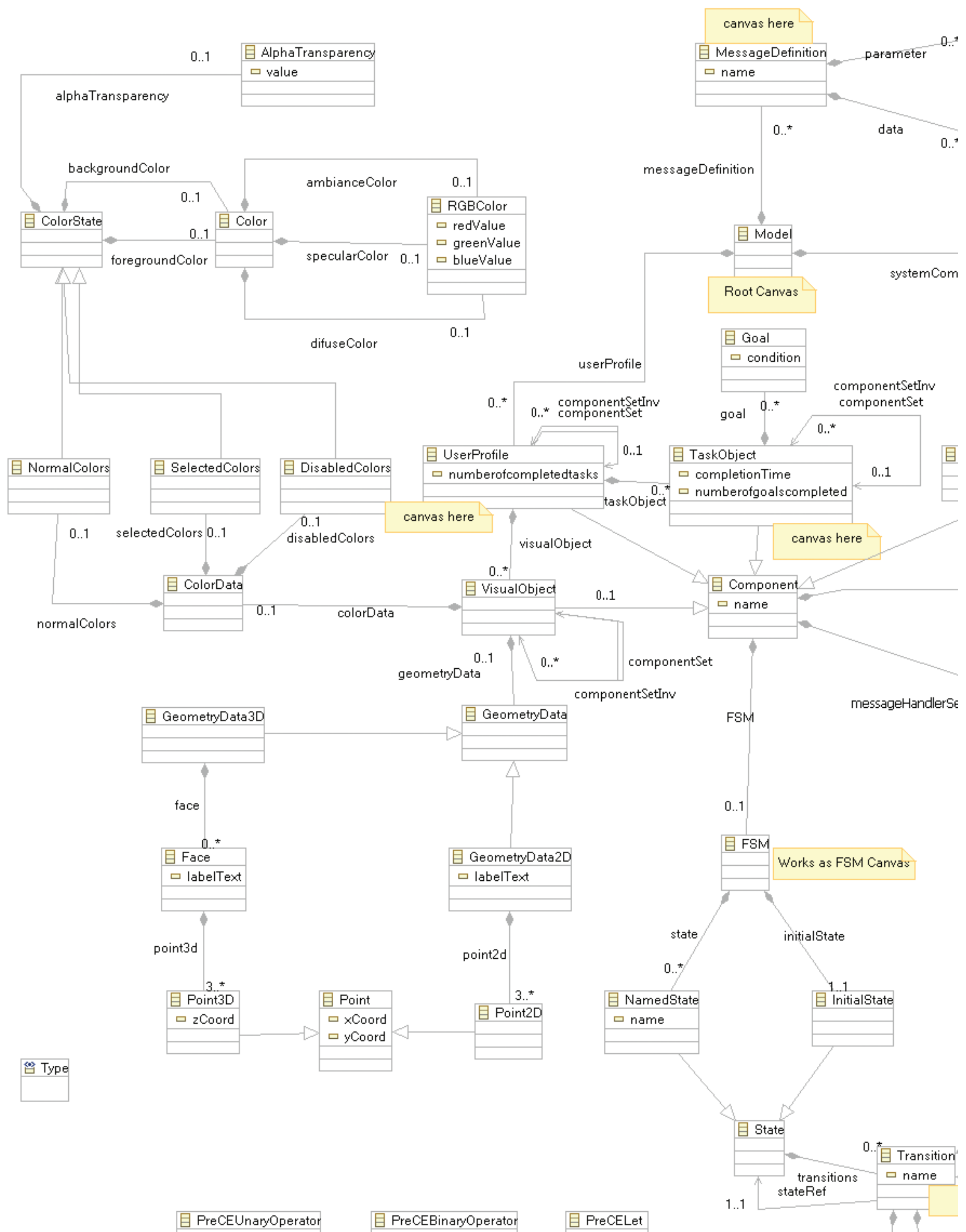

Figure A.2 Upper left section of the (H)ALL Meta-Model

PreCMEGetMessageParameter
field
PreCMEUnaryOperator
operatorname
PreCMEBinaryOperato
operatorname
Parameter
name
type
arameter
0..*
PreCMEGetMessageData
field
expression
leftexpression
rightexpression
1..1
1..1
1..1
Data
name
type
initValue
data
0..*
0..*
PreCMEDomainPropertyGet
name
PreConditionMessageExpressionElement
PreCMELiteral
value
0..*
1..1
in
1..1
initializat
PreConditionSet
systemComponent
0..*
PreConditionMessageExpression
PreCMEVarRef
name
type
Works as expressionCanvas
0..1
componentSetInv
nentSetInv
componentSet
onentSet
data
PreCondition
0..*
SystemComponent
0..1
0..1
canvas
canvas here
MessageTransition
name
InitialMessageState
PosCondition
0..*
1..1
stateRef
ActionMessage
initialMessageState
transitions
1..1
MessageState
isEnd
isContinue
MessageHandler
name
NamedMessageState
name
ssageHandlerSet
messageState
0..*
0..*
Works as Message FSM Canvas
0..1
Canvas
MessageNotification
tate
TriggerExpression
TriggerExpressionSet
TriggerExpressionElement
String
DomainEventFired
0..1
canvas
Trigger
0..1
Transition
name
Action
AEBinaryOperator
operatorname
AEUnaryOperator
operatorname
canvas
expression

Figure A.3 Upper middle left section of the (H)ALL Meta-Model

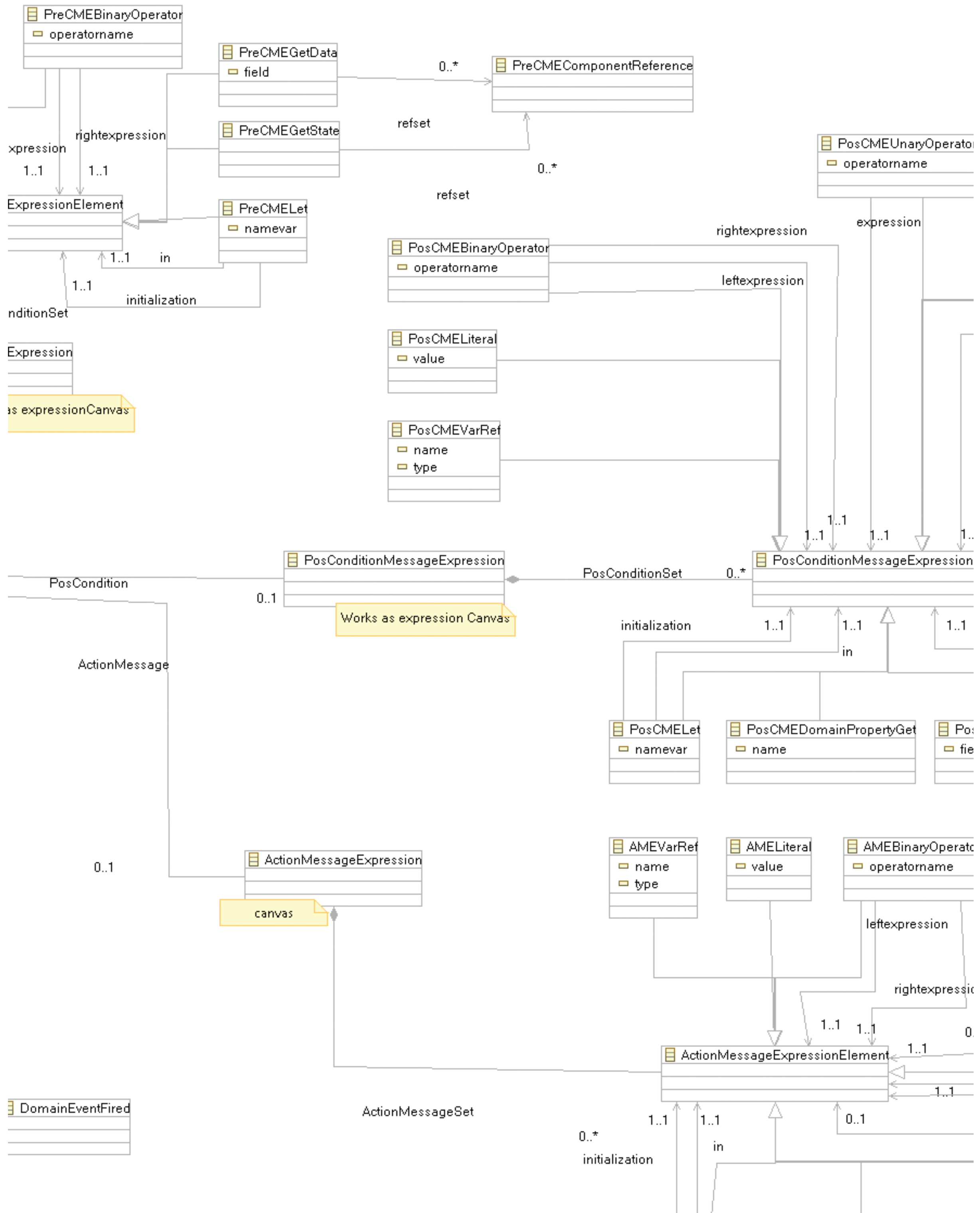

Figure A.4 Upper middle right section of the (H)ALL Meta-Model

Figure A.5 Upper right section of the (H)ALL Meta-Model

State
Transition
name
transitions
stateRef
0..*
1..1
car
PreCEUnaryOperator
operatorname
PreCEBinaryOperator
operatorname
PreCELet
namevar
in
initialization
expression
leftexpression
rightexpression
1..1
1..1
1..1
PreCondition
PreCEVarRef
name
type
PreConditionExpressionElement
PreConditionSet
0..*
0..1
PreConditionExpression
Works as expression Canvas
PreCELiteral
value
PosCon
PreCEDomainPropertyGet
name
PreCEGetState
PreCEGetData
field
stateRefSet
dataRefSet
0..*
0..*
0..1
PreCEComponentReference
PosConditionE
PosCEUnaryOperator
operatorname
expression
1..1
PosConditionE
PosCEBinaryOperator
operatorname
1..1
1..1
1..1
1..1
rightexpression
leftexpression
initialization
in
PosCELet
namevar
PosCEGetState
PosCEGetData
field
dataRefSet
stateRefSet
0..*
PosCECo

Figure A.6 Lower left section of the (H)ALL Meta-Model

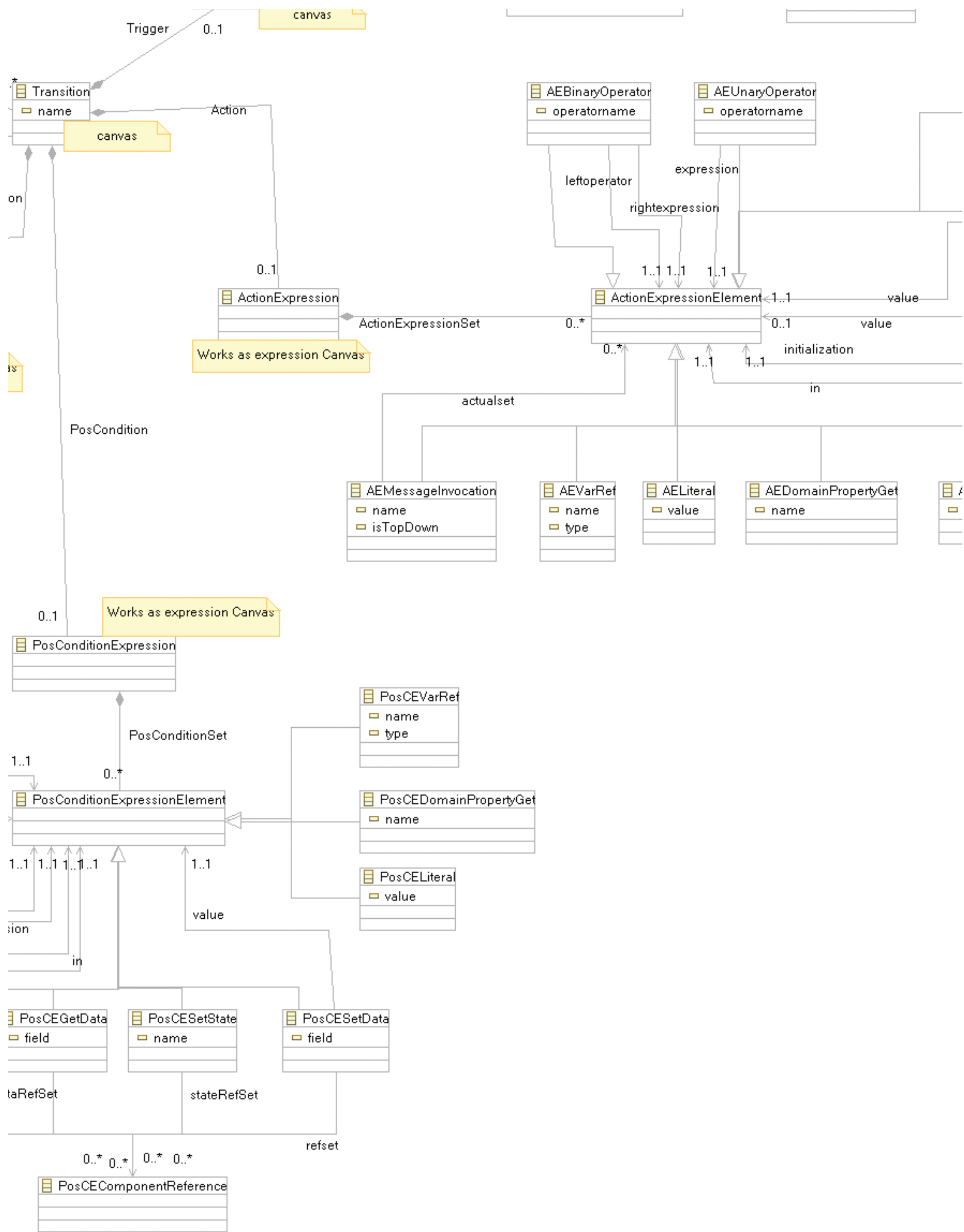

Figure A.7 Lower middle left section of the (H)ALL Meta-Model

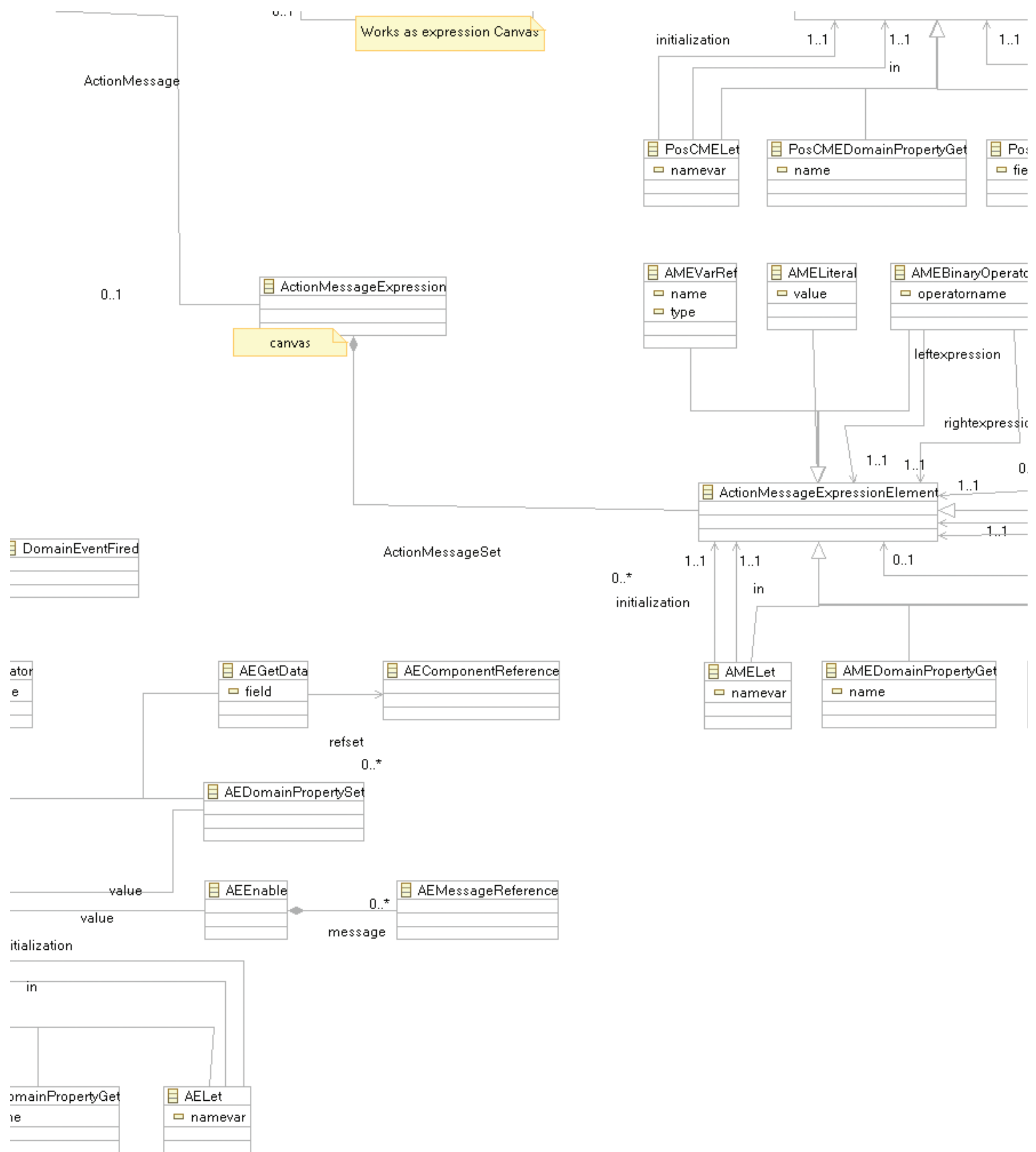

Figure A.8 Lower middle right section of the (H)ALL Meta-Model

Figure A.9 Lower right section of the (H)ALL Meta-Model

B. CO-OPN Metamodel Diagrams

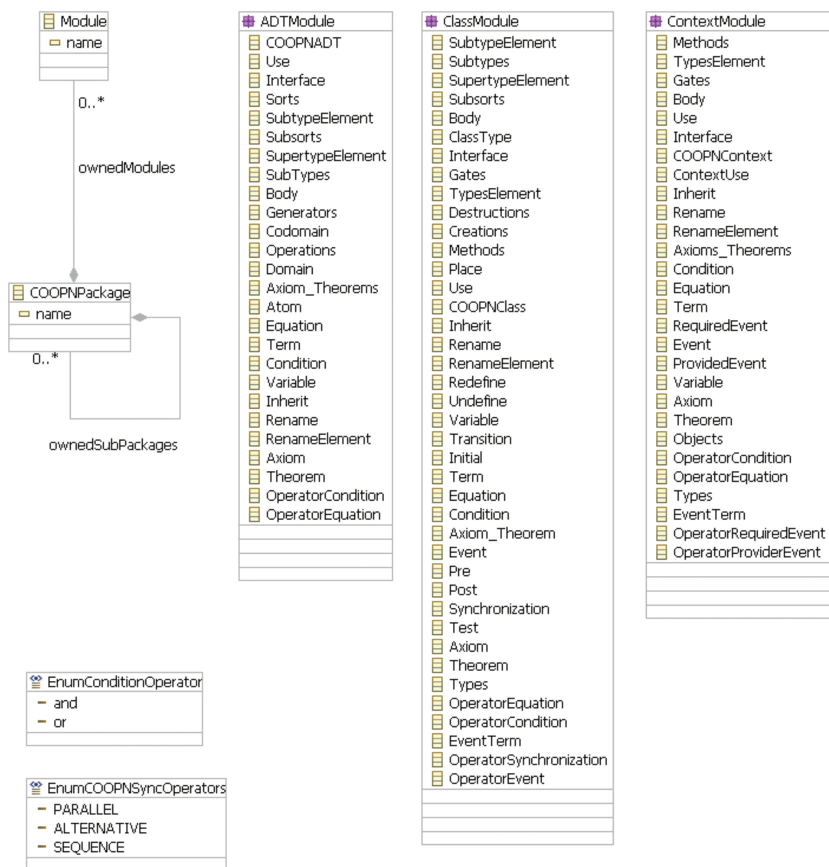

Figure B.1 CO-OPN Meta-model

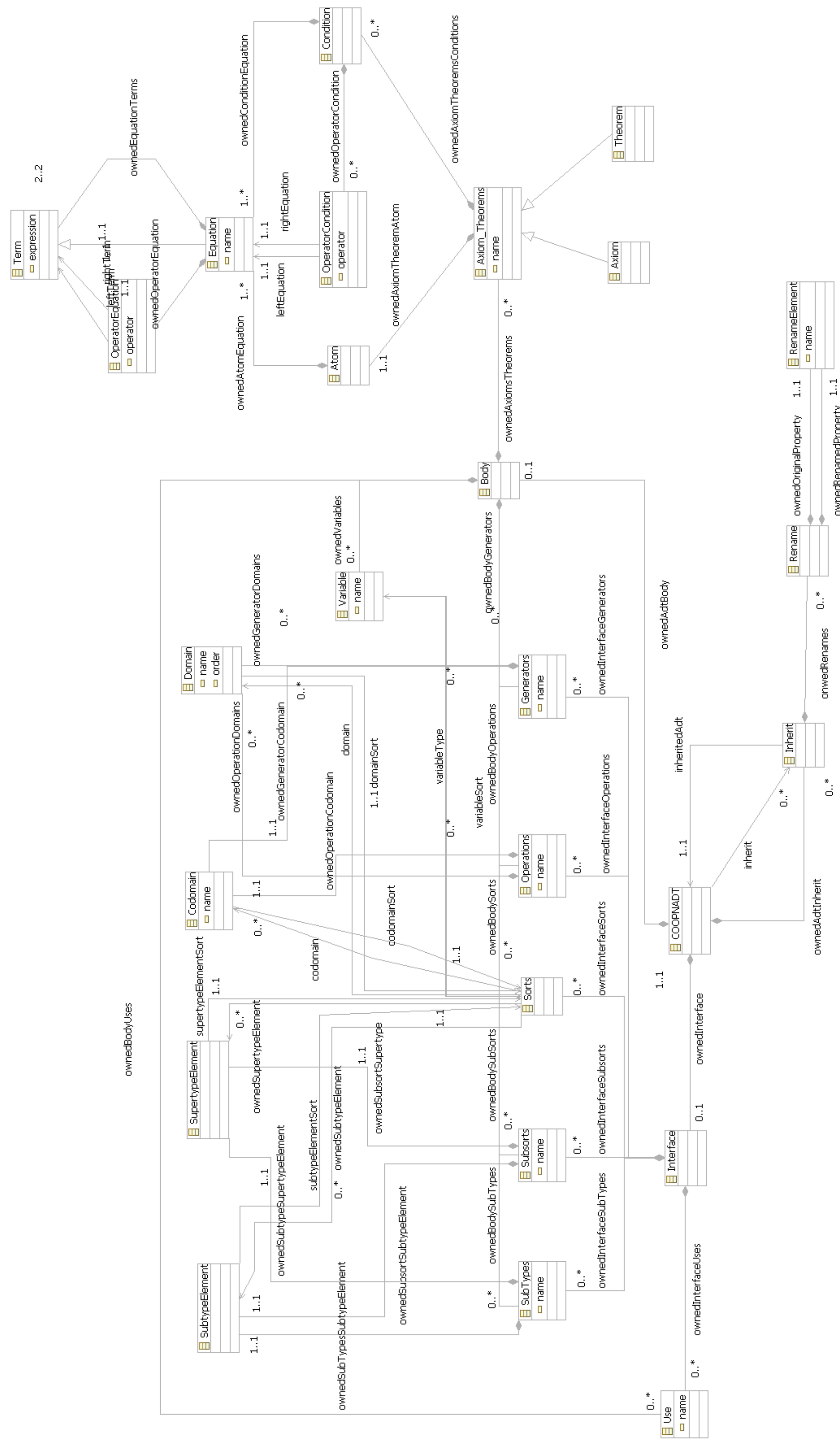

Figure B.2 CO-OPN ADT Module Meta-module

Figure B.3 CO-OPN Class Module Meta-module

Figure B.4 CO-OPN Context Module Meta-module

C . Editor Dependencies

In this appendix we present the main file variables affections for the chaining of the several produced editors into one single tool, as presented in figure 5.5. The variables affections not present in this appendix, are approached in the same manner as the presented here. The annotation of these variable affections where essential for an efficient production cycle.

The variables are presented in the following manner. Each editor has its own section. Within that section, every file with variables that need to be altered in the production of each editor are placed in a sub section. On every sub section a listing of the navigation on the file is presented in the tree like manner of the original files, where the last level viewed has the variables of the properties tab presented in **bold**, followed by the variable value to be set. The variables with their values in *italic* where automatically altered with the alteration of the other variables, but never the less should be taken into account.

C.1 Root Editor

C.1.1 Hall.gmfgen file variables

1. Gen Editor Generator
2. **Diagram File Extention** hall_diagram
3. **Domain File Extention** HALL
4. **Model ID** HALL
5. **Domain Gen Model** HALL
6. **Package Name Prefix** HALL.diagram
7. Gen Plugin
8. **ID** HALL.diagram
9. **Name** HALL Plugin
10. **Activator Class Name** HALLDiagramEditorPlugin
11. Gen Diagram ModelEditPart
12. Gen Top Level Node UserProfileEditPart
13. Open Diagram Behaviour
14. **Diagram Kind** UserProfile
15. **Editor Id** UserProfile.diagram.part.UserProfileDiagramEditorID
16. **Edit Policy Class Name** UserProfileDiagramEditPolicy
17. Gen Top Level Node MessageDefinitionEditPart
18. Open Diagram Behaviour
19. **Diagram Kind** MessageDefinition
20. **Editor Id** MessageDefinition.diagram.part.MessageDefinitionDiagramEditorID
21. **Edit Policy Class Name** MessageDefinitionDiagramEditPolicy
22. Gen Top Level Node SystemComponentEditPart
23. Open Diagram Behaviour

24. **Diagram Kind** SystemComponent
25. **Editor Id** SystemComponent.diagram.part.SystemComponentDiagramEditorID
26. **Edit Policy Class Name** SystemComponentDiagramEditPolicy

C.1.2 Hall.gmfmap file variables

1. Platform
2. Mapping
3. Top Node Reference<userProfile>
4. Node Maping
5. **Misc->Related Diagrams** Canvas maping(userProfile.gmfmap)
6. Top Node Reference<messageDefinition>
7. Node Maping
8. **Misc->Related Diagrams** Canvas maping(MessageDefinition.gmfmap)
9. Top Node Reference<systemComponent>
10. Node Maping
11. **Misc->Related Diagrams** Canvas maping(SystemComponent.gmfmap)

C.2 UserProfile Editor

C.2.1 UserProfile.gmfgen file variables

1. Gen Editor Generator
2. **Diagram File Extention** *hall_diagram*
3. **Domain File Extention** *HALL*
4. **Model ID** UserProfile
5. **Domain Gen Model** UserProfile
6. **Package Name Prefix** UserProfile.diagram
7. Gen Plugin
8. **ID** UserProfile.diagram
9. **Name** *UserProfile Plugin*
10. **Activator Class Name** *UserProfileDiagramEditorPlugin*
11. Gen Diagram ModelEditPart
12. Gen Top Level Node MessageHandlerEditPart
13. Open Diagram Behaviour
14. **Diagram Kind** MessageHandler
15. **Editor Id** MessageHandler.diagram.part.MessageHandlerDiagramEditorID
16. **Edit Policy Class Name** MessageHandlerDiagramEditPolicy
17. Gen Top Level Node TaskObjectEditPart
18. Open Diagram Behaviour
19. **Diagram Kind** TaskObject
20. **Editor Id** TaskObject.diagram.part.TaskObjectDiagramEditorID
21. **Edit Policy Class Name** TaskObjectDiagramEditPolicy
22. Gen Top Level Node VisualObjectEditPart
23. Open Diagram Behaviour

24. **Diagram Kind** VisualObject
25. **Editor Id** VisualObject.diagram.part.VisualObjectDiagramEditorID
26. **Edit Policy Class Name** VisualObjectDiagramEditPolicy
27. Gen Top Level Node FSMEditPart
28. Open Diagram Behaviour
29. **Diagram Kind** FSM
30. **Editor Id** FSM.diagram.part.FSMDiagramEditorID
31. **Edit Policy Class Name** FSMDiagramEditPolicy

C.2.2 UserProfile.genmodel file variables

1. UserProfile
2. **Model Name** UserProfile

C.2.3 UserProfile.gmfmap file variables

1. Platform
2. Mapping
3. Top Node Reference<fsm>
4. Node Maping
5. **Misc->Related Diagrams** Canvas maping(fsm.gmfmap)
6. Top Node Reference<fsm>
7. Node Maping
8. **Misc->RelatedDiagram** Canvas maping(messageHandler.gmfmap)

D. ATL File Listing

In this appendix we present the listing of ATL rule files. Each file corresponds to a different execution layer. Code lines in *italic* preceded by a – are comments and do not influence the execution process. Rules that are similar to others are referenced as so to the complete rule in comment and the specific code is omitted.

D.1 Declarative01.atl

```
1  module Declarative01;
2  create coopnModel : COOPN from hallModel : HALL;
3
4  rule base{
5     from
6        s : HALL!Model
7     to
8        t : COOPN!COOPNPackage(
9           name <- 'HALLModelPackage' ,
10          ownedModules <- Set{base,gc,gu,gsc,gvo,gto}
11       ),
12       base : COOPN!COOPNContext(
13          name <- 'BaseContext',
14          ownedBody <- baseBody
15       ),
16       baseBody : COOPN!Body(
17       ),
18       gu : COOPN!COOPNContext(
19          name <- 'GenericUserProfile',
20          ownedInherits <- igu
21       ),
22       igu : COOPN!Inherit(
23          inheritedContext <- gc
24       ),
25       gsc : COOPN!COOPNContext(
26          name <- 'GenericSystemComponent',
27          ownedInherits <- igsc
28       ),
29       igsc : COOPN!Inherit(
30          inheritedContext <- gc
31       ),
32       gvo : COOPN!COOPNContext(
33          name <- 'GenericVisualObject',
34          ownedInherits <- igvo
35       ),
36       igvo : COOPN!Inherit(
37          inheritedContext <- gc
38       ),
39       gto : COOPN!COOPNContext(
40          name <- 'GenericTaskObject',
```

```
41            ownedInherits <- igto
42        ),
43        igto : COOPN!Inherit(
44            inheritedContext <- gc
45        ),
46        gc : COOPN!COOPNContext(
47            name <- 'GenericComponent'
48        )
49 }
```

D.2 Declarative02.atl

```
1  module Declarative02;
2  create coopnModel : COOPN refining hallModel : HALL, preCoopnModel :
      COOPN;
3
4   rule packageRefactor{
5     from
6         s : COOPN!COOPNPackage
7     to
8         t : COOPN!COOPNPackage(
9             name <- s.name ,
10            ownedModules <- Set{s.ownedModules}
11                ->union(Set{HALL!UserProfile.allInstancesFrom('hallModel
                      ')}->flatten() )
12                ->union(Set{HALL!SystemComponent.allInstancesFrom('
                      hallModel')}->flatten() )
13                ->union(Set{HALL!VisualObject.allInstancesFrom('hallModel
                      ')}->flatten() )
14                ->union(Set{HALL!TaskObject.allInstancesFrom('hallModel')
                      }->flatten() )
15        )
16 }
17
18 rule inherit{
19    from
20        s : COOPN!Inherit
21    to
22        t : COOPN!Inherit(
23        )
24 }
25
26 rule userprofiletocontext{
27    from
28        h : HALL!UserProfile
29    to
30        c : COOPN!COOPNContext(
31            name <- 'UserProfile' + h.name,
32            ownedBody <- b,
33            ownedInherits <- d
34        ),
35        b : COOPN!Body (
```

```
36          ),
37          d : COOPN!Inherit (
38              inheritedContext <- COOPN!COOPNContext.allInstancesFrom('
                    preCoopnModel')->any(e | e.name = 'GenericUserProfile')
39          )
40  }
41
42  rule systemcoponent2context{
43      from
44          h : HALL!SystemComponent
45      to
46          -- similar to rule userprofiletocontext where UserProfile is
                replaced by SystemComponent
47  }
48
49  rule visualobject2context{
50      from
51          h : HALL!VisualObject
52      to
53          -- similar to rule userprofiletocontext where UserProfile is
                replaced by VisualObject
54  }
55
56  rule taskobject2context{
57      from
58          h : HALL!TaskObject
59      to
60          -- similar to rule userprofiletocontext where UserProfile is
                replaced by TaskObject
61  }
```

D.3 Declarative03.atl

```
1  module Declarative03;
2  create coopnModel : COOPN refining hallModel : HALL, preCoopnModel :
      COOPN;
3
4  rule packageRefactor{
5      from
6          s : COOPN!COOPNPackage
7      to
8          t : COOPN!COOPNPackage(
9              name <- s.name ,
10             ownedModules <- Set{s.ownedModules}
11         )
12 }
13
14 rule inherit{
15     from
16         s : COOPN!"COOPNMetamodel::ContextModule::Inherit"
17     to
18         t : COOPN!"COOPNMetamodel::ContextModule::Inherit"(
19         )
20 }
21
22 rule baseBodyRefactor{
23     from
24         s : COOPN!"COOPNMetamodel::ContextModule::Body"
25         (s.contextContainsBody.name = 'BaseContext')
26     to
27         t : COOPN!"COOPNMetamodel::ContextModule::Body"(
28                 ownedContextUses <- s.ownedContextUses
29                     -> union(HALL!UserProfile.allInstancesFrom('hallModel
                           '))
30                     -> union(HALL!SystemComponent.allInstancesFrom('
                           hallModel'))
31                     -> union(HALL!VisualObject.allInstancesFrom('hallModel
                           '))
32                     -> union(HALL!TaskObject.allInstancesFrom('hallModel')
                           )
33         )
34 }
35
```

```
36 rule userprofiletocontextuse{
37     from
38         h : HALL!UserProfile
39     to
40         c : COOPN!ContextUse(
41             usedContext <- COOPN!COOPNContext.allInstancesFrom('
                   preCoopnModel')->any( e | e.name = 'UserProfile' + h.
                   name )
42         )
43 }
44
45 rule systemcomponenttocontextuse{
46     from
47         h : HALL!SystemComponent
48     to
49         -- similar to rule userprofiletocontextuse where UserProfile is
                replaced by SystemComponent
50 }
51
52 rule visualobjecttocontextuse{
53     from
54         h : HALL!VisualObject
55     to
56         -- similar to rule userprofiletocontextuse where UserProfile is
                replaced by VisualObject
57 }
58
59 rule taskobjecttocontextuse{
60     from
61         h : HALL!TaskObject
62     to
63         -- similar to rule userprofiletocontextuse where UserProfile is
                replaced by TaskObject
64 }
```

D.4 FSM01.atl

```
1  module FSM01;
2  create coopnModel : COOPN refining hallModel : HALL, preCoopnModel :
      COOPN, coopnTypes : COOPN;
3
4  rule baseRefactor{
5      from
6          s : COOPN!COOPNPackage
7          (s.name = 'HALLModelPackage')
8      to
9          t : COOPN!COOPNPackage(
10             name <- s.name ,
11             ownedModules <- Set{s.ownedModules}
12                 ->union(Set{HALL!FSM.allInstancesFrom('hallModel')}->
                       flatten())
13                 ->union(Set{COOPN!COOPNADT.allInstancesFrom('coopnTypes')
                       }->flatten() )
14         )
15 }
16
17 rule inherit{
18     from
19         s : COOPN!Inherit
20     to
21         t : COOPN!Inherit(
22         )
23 }
24
25 rule sorts{
26     from
27         s : COOPN!"COOPNMetamodel::ADTModule::Sorts"
28     to
29         t : COOPN!"COOPNMetamodel::ADTModule::Sorts"(
30             name <- s.name
31         )
32 }
33
34 rule contextBodyRefactor{
35     from
36         s : COOPN!"COOPNMetamodel::ContextModule::Body"
37         (HALL!FSM.allInstancesFrom('hallModel')
```

```
38                ->exists(i|
39                    i.FSMInv.oclType().toString()
40                        .split('!').at(2) + i.FSMInv.name = s.
                            contextContainsBody.name))
41      to
42          t : COOPN!"COOPNMetamodel::ContextModule::Body"(
43              ownedContextUses <- s.ownedContextUses,
44              ownedBodyUses <- u,
45              ownedObjects <- o
46          ),
47          u : COOPN!"COOPNMetamodel::ContextModule::Use"(
48              name <- s.contextContainsBody.name + 'FSM',
49              usedModuleForContext <- HALL!FSM.allInstancesFrom('hallModel
                    ')->any(e| (e.FSMInv.oclType().toString()
50                        .split('!').at(2) + e.FSMInv.name)= s.
                            contextContainsBody.name)
51          ),
52          o : COOPN!"COOPNMetamodel::ContextModule::Objects"(
53              name <- 'FSMObject'
54          )
55  }
56
57  helper context COOPN!"COOPNMetamodel::ContextModule::Body" def:
        getObjectType(): COOPN!"COOPNMetamodel::ClassModule::ClassType" =
58          self.contextContainsBody.name
59      ;
60
61  rule FSM2Class{
62      from
63          h : HALL!FSM
64      to
65          c : COOPN!"COOPNMetamodel::ClassModule::COOPNClass"(
66              name <- h.FSMInv.oclType().toString()
67                        .split('!').at(2) + h.FSMInv.name + 'FSM' ,
68              ownedInterface <- int,
69              ownedBody <- b
70          ),
71          b : COOPN!"COOPNMetamodel::ClassModule::Body"(
72              ownedPlaces <- h.getStates(),
73              ownedAxiomTheorems <- Set{HALL!Transition.allInstancesFrom('
                    hallModel')->select(t|t.source.fsm = h or t.stateRef.fsm
                    = h)}, -- tenho que corrigir isto
74              ownedInitial <- init,
```

```
75             ownedVariables <- var
76         ),
77         int : COOPN!"COOPNMetamodel::ClassModule::Interface"(
78             ownedInterfaceClassTypes <- Set{ct}
79         ),
80         ct : COOPN!"COOPNMetamodel::ClassModule::ClassType"(
81             name <- c.name + 'Type',
82             order <- 1
83         ),
84         init : COOPN!"COOPNMetamodel::ClassModule::Initial"(
85             ownedInitialTerms <- Set{oit}
86         ),
87         oit : COOPN!Term(
88             expression <- 'InitialPlace @'
89         ),
90         var : COOPN!"COOPNMetamodel::ClassModule::Variable"(
91             name <- 'this',
92             variableType <- ct
93         )
94 }
95
96 helper context HALL!FSM def : getTransitions() : Set (COOPN!"
       COOPNMetamodel::ClassModule::Axiom") =
97         self.state
98     ;
99
100 helper context HALL!FSM def : getStates() : Set (COOPN!"
       COOPNMetamodel::ClassModule::Place") =
101         Set{self.initialState} -> union(Set{self.state})
102     ;
103
104 rule NamedState2Place{
105     from
106         h : HALL!NamedState
107     to
108         c : COOPN!"COOPNMetamodel::ClassModule::Place"(
109             name <- h.name,
110             ownedPlaceTypeElements <- o
111         ),
112         o : COOPN!"COOPNMetamodel::ClassModule::TypesElement"(
113             order <- 1,
114             typeElementType <- COOPN!Sorts.allInstancesFrom('coopnTypes
                 ') -> any( e | e.name = 'blacktoken' )
```

```
115         )
116 }
117
118 rule InitialState2Place{
119     from
120         h : HALL!InitialState
121     to
122         -- similar to rule NamedState2Place where h.name is replaced by
                'InitialPlace'
123 }
124
125 rule Transition2Axiom{
126     from
127         h : HALL!Transition
128     to
129         c : COOPN!"COOPNMetamodel::ClassModule::Axiom"(
130             name <- h.name,
131             ownedPre <- sr,
132             ownedPost <- tr,
133             ownedAxiomTheoremSychronisation <- synch,
134             ownedEvent <- evt,
135             ownedCondition <- cond
136         ),
137         sr : COOPN!"COOPNMetamodel::ClassModule::Pre"(
138             ownedPreTerm <- srt
139         ),
140         srt : COOPN!"COOPNMetamodel::ClassModule::Term"(
141             expression <- if(h.source.oclIsTypeOf(HALL!InitialState) )
                    then
142                         'InitialPlace @'
143                     else
144                         h.source.name + ' @'
145                     endif
146         ),
147         tr : COOPN!"COOPNMetamodel::ClassModule::Post"(
148             ownedPostTerm <- trt
149         ),
150         trt : COOPN!"COOPNMetamodel::ClassModule::Term"(
151             expression <- if(h.stateRef.oclIsTypeOf(HALL!InitialState) )
                    then
152                         'InitialPlace @'
153                     else
154                         h.stateRef.name + ' @'
```

```
155                     endif
156         ),
157         synch : COOPN!"COOPNMetamodel::ClassModule::Synchronization"(
158             ownedEventTerms <- Set{syet}
159         ),
160         syet : COOPN!EventTerm(
161             expression <- 'this.fire' + c.name + 'post'
162         ),
163         evt : COOPN!"COOPNMetamodel::ClassModule::Event"(
164             owenedEventTerm <- Set{evtet}
165         ),
166         evtet : COOPN!"COOPNMetamodel::ClassModule::EventTerm"(
167             expression <- 'fire' + c.name + 'pre'
168         ),
169         cond : COOPN!"COOPNMetamodel::ClassModule::Condition"(
170             ownedEquations <- eq
171         ),
172         eq : COOPN!"COOPNMetamodel::ClassModule::Equation"(
173             ownedOperatorEquation <- opeq,
174             ownedEquationTerms <- Set{lterm,rterm}
175         ),
176         lterm : COOPN!"COOPNMetamodel::ClassModule::Term"(
177             expression <- 'this'
178         ),
179         rterm : COOPN!"COOPNMetamodel::ClassModule::Term"(
180             expression <- 'Self'
181         ),
182         opeq : COOPN!"COOPNMetamodel::ClassModule::OperatorEquation"(
183             leftTerm <- lterm,
184             rightTerm <- rterm,
185             operator <- '='
186         )
187 }
```

D.5 FSM02.atl

```
1  module FSM02;
2  create coopnModel : COOPN refining preCoopnModel : COOPN, hallModel
     : HALL;
3
4  rule PackageRefactor{
5      from
6          s : COOPN!COOPNPackage
7      to
8          t : COOPN!COOPNPackage(
9              name <- s.name ,
10             ownedModules <- s.ownedModules
11         )
12 }
13
14 rule inherit{
15     from
16         s : COOPN!Inherit
17     to
18         t : COOPN!Inherit(
19         )
20 }
21
22 rule sorts{
23     from
24         s : COOPN!"COOPNMetamodel::ADTModule::Sorts"
25     to
26         t : COOPN!"COOPNMetamodel::ADTModule::Sorts"(
27             name <- s.name
28         )
29 }
30
31 rule ObjectClassTypeSet{
32     from
33         s : COOPN!"COOPNMetamodel::ContextModule::Objects"
34     to
35         t : COOPN!"COOPNMetamodel::ContextModule::Objects"(
36             name <- s.name ,
37             objectType <- COOPN!"COOPNMetamodel::ClassModule::ClassType
                 ".allInstancesFrom('preCoopnModel')->any(e| e.name = s.
                 bodyContainsObjects.contextContainsBody.name + 'FSMType')
```

```
38          )
39  }
40
41  rule FSMClassRefactor{
42      from
43          s : COOPN!"COOPNMetamodel::ClassModule::Body"
44      to
45          t : COOPN!"COOPNMetamodel::ClassModule::Body"(
46              ownedPlaces <- s.ownedPlaces,
47              ownedInitial <- s.ownedInitial,
48              ownedAxiomTheorems <- s.ownedAxiomTheorems,
49              ownedBodyMethods <- HALL!Transition.allInstancesFrom('
                    hallModel')->select( e |
50                  s.coopnClassContainsBody.name.endsWith(e.source.fsm.
                        FSMInv.name + 'FSM') )
51          )
52
53  }
54
55  rule Transition2Method{
56      from
57          h : HALL!Transition
58      to
59          c : COOPN!"COOPNMetamodel::ClassModule::Methods"(
60              name <- 'fire' + h.name + 'pre'
61          )
62  }
```

D.6 FSM03.atl

```
1  module FSM03;
2  create coopnModel : COOPN refining preCoopnModel : COOPN, hallModel
      : HALL;
3
4  rule PackageRefactor{
5      from
6          s : COOPN!COOPNPackage
7      to
8          t : COOPN!COOPNPackage(
9              name <- s.name ,
10             ownedModules <- Set {s.ownedModules}
11         )
12 }
13
14 rule inherit{
15     from
16         s : COOPN!Inherit
17     to
18         t : COOPN!Inherit(
19         )
20 }
21
22 rule object{
23     from
24         s : COOPN!"COOPNMetamodel::ContextModule::Objects"
25     to
26         t : COOPN!"COOPNMetamodel::ContextModule::Objects"(
27             name <- s.name
28         )
29 }
30
31 rule sorts{
32     from
33         s : COOPN!"COOPNMetamodel::ADTModule::Sorts"
34     to
35         t : COOPN!"COOPNMetamodel::ADTModule::Sorts"(
36             name <- s.name
37         )
38 }
39
```

```
40 rule FSMClassInterfaceRefactor{
41     from
42         s : COOPN!"COOPNMetamodel::ClassModule::Interface"
43         (s.coopnClassContainsInterface.name.endsWith('FSM'))
44     to
45         t : COOPN!"COOPNMetamodel::ClassModule::Interface"(
46             ownedInterfaceClassTypes <- s.ownedInterfaceClassTypes,
47             ownedInterfaceGates <- Set{HALL!Transition.allInstancesFrom
                   ('hallModel')->select( e |
48                 s.coopnClassContainsInterface.name.endsWith(e.source.fsm.
                       FSMInv.name + 'FSM') )}
49         )
50 
51 }
52 
53 rule Transition2Gates{
54     from
55         h : HALL!Transition
56     to
57         c : COOPN!"COOPNMetamodel::ClassModule::Gates"(
58             name <- 'fire' + h.name + 'post'
59         )
60 }
```

D.7 mfsm00.atl

```
1  module MFSM00;
2  create coopnModel : COOPN refining hallModel : HALL, preCoopnModel :
     COOPN;
3
4  rule baseRefactor{
5     from
6        s : COOPN!COOPNPackage
7     to
8        t : COOPN!COOPNPackage(
9           name <- s.name ,
10          ownedModules <- s.ownedModules ->union(
11             Set{
12                HALL!MessageHandler.allInstancesFrom('hallModel')
13             }
14          )
15       )
16 }
17
18 rule inherit{
19    from
20       s : COOPN!Inherit
21    to
22       t : COOPN!Inherit(
23       )
24 }
25
26  rule object{
27    from
28       s : COOPN!"COOPNMetamodel::ContextModule::Objects"
29    to
30       t : COOPN!"COOPNMetamodel::ContextModule::Objects"(
31          name <- s.name
32       )
33 }
34
35 rule sorts{
36    from
37       s : COOPN!"COOPNMetamodel::ADTModule::Sorts"
38    to
39       t : COOPN!"COOPNMetamodel::ADTModule::Sorts"(
```

```
40              name <- s.name
41          )
42  }
43
44  rule contextBodyRefactor{
45      from
46          s : COOPN!"COOPNMetamodel::ContextModule::Body"
47          (HALL!MessageHandler.allInstancesFrom('hallModel')
48                  ->exists(i| let ContextType : String = i.
                        messageHandlerSetInv.oclType().toString()
49                      in ( ContextType.split('!').at(2) + i.
                            messageHandlerSetInv.name ) = s.contextContainsBody
                            .name))
50      to
51          t : COOPN!"COOPNMetamodel::ContextModule::Body"(
52              ownedContextUses <- s.ownedContextUses,
53              ownedBodyUses <- s.ownedBodyUses ->union(Set{HALL!
                    MessageHandler.allInstancesFrom('hallModel') -> select (e
                    |
54                  let ContextType : String = e.messageHandlerSetInv.oclType
                        ().toString()
55                  in ( ContextType.split('!').at(2) + e.
                        messageHandlerSetInv.name ) = s.contextContainsBody.
                        name
56                  )}),
57              ownedObjects <- s.ownedObjects
58          )
59  }
60
61  rule MessageHandler2Use{
62      from
63          h : HALL!MessageHandler
64      to
65          c : COOPN!"COOPNMetamodel::ContextModule::Use"(
66              name <- h.messageHandlerSetInv.oclType().toString().split
                    ('!').at(2) + h.messageHandlerSetInv.name + h.name + '
                    MessageHandler'
67          )
68  }
```

D.8 mfsm01.atl

```
1  module MFSM01;
2  create coopnModel : COOPN refining hallModel : HALL, preCoopnModel :
      COOPN;
3
4  rule baseRefactor{
5     from
6        s : COOPN!COOPNPackage
7     to
8        t : COOPN!COOPNPackage(
9           name <- s.name ,
10          ownedModules <- s.ownedModules ->union(
11             Set{
12                HALL!MessageHandler.allInstancesFrom('hallModel')
13             }
14          )
15       )
16 }
17
18 rule inherit{
19    from
20       s : COOPN!Inherit
21    to
22       t : COOPN!Inherit(
23       )
24 }
25
26  rule object{
27    from
28       s : COOPN!"COOPNMetamodel::ContextModule::Objects"
29    to
30       t : COOPN!"COOPNMetamodel::ContextModule::Objects"(
31          name <- s.name
32       )
33 }
34
35 rule sorts{
36    from
37       s : COOPN!"COOPNMetamodel::ADTModule::Sorts"
38    to
39       t : COOPN!"COOPNMetamodel::ADTModule::Sorts"(
```

```
40              name <- s.name
41          )
42  }
43
44  helper context COOPN!"COOPNMetamodel::ContextModule::Body" def:
        getObjectType(): COOPN!"COOPNMetamodel::ClassModule::ClassType" =
45          self.contextContainsBody.name
46      ;
47
48  rule useRefactor{
49      from
50          s : COOPN!"COOPNMetamodel::ContextModule::Use"
51          (s.name.endsWith('MessageHandler'))
52
53      to
54          t : COOPN!"COOPNMetamodel::ContextModule::Use"(
55              name <- s.name,
56              usedModuleForContext <- HALL!MessageHandler.allInstancesFrom
                    ('hallModel') -> any(h|
57                  let ContextType : String = h.messageHandlerSetInv.oclType
                        ().toString()
58                  in ( ( ContextType.split('!').at(2) + h.
                        messageHandlerSetInv.name + h.name + 'MessageHandler')
                         = s.name )
59              )
60          )
61  }
62
63  rule MessageHandler2Class{
64      from
65          h : HALL!MessageHandler
66      to
67          c : COOPN!"COOPNMetamodel::ClassModule::COOPNClass"(
68              name <- h.messageHandlerSetInv.oclType().toString()
69                          .split('!').at(2) + h.messageHandlerSetInv.name + h
                                .name + 'MessageHandler' ,
70              ownedInterface <- int ,
71              ownedBody <- b
72          ),
73          b : COOPN!"COOPNMetamodel::ClassModule::Body"(
74              ownedPlaces <- h.getMessageStates(),
75              ownedAxiomTheorems <- Set{HALL!MessageTransition.
                    allInstancesFrom('hallModel') ->select(t|
```

```
76              if t.transitionsInvMessageState.oclType().toString().
                    endsWith('InitialMessageState')
77              then t.transitionsInvMessageState.initialMessageStateInv
                    = h
78              else t.transitionsInvMessageState.messageStateInv = h
79              endif )},
80          ownedInitial <- init,
81          ownedVariables <- var
82      ),
83      int : COOPN!"COOPNMetamodel::ClassModule::Interface"(
84          ownedInterfaceClassTypes <- Set{ct}
85      ),
86      ct : COOPN!"COOPNMetamodel::ClassModule::ClassType"(
87          name <- c.name + 'Type',
88          order <- 1
89      ),
90      init : COOPN!"COOPNMetamodel::ClassModule::Initial"(
91          ownedInitialTerms <- Set{oit}
92      ),
93      oit : COOPN!Term(
94          expression <- 'InitialPlace @'
95      ),
96      var : COOPN!"COOPNMetamodel::ClassModule::Variable"(
97          name <- 'this',
98          variableType <- ct
99      )
100 }
101
102 helper context HALL!MessageHandler def : getMessageTransitions() :
        Set (COOPN!"COOPNMetamodel::ClassModule::Axiom") =
103         self.state
104     ;
105
106 helper context HALL!MessageHandler def : getMessageStates() : Set (
        COOPN!"COOPNMetamodel::ClassModule::Place") =
107         Set{self.initialMessageState} -> union(Set{self.messageState})
108     ;
109
110 rule NamedMessageState2Place{
111     from
112         h : HALL!NamedMessageState
113     to
114         c : COOPN!"COOPNMetamodel::ClassModule::Place"(
```

```
115            name <- h.name,
116            ownedPlaceTypeElements <- o
117        ),
118        o : COOPN!"COOPNMetamodel::ClassModule::TypesElement"(
119            order <- 1,
120            typeElementType <- COOPN!Sorts.allInstancesFrom('
                 preCoopnModel') -> any( e | e.name = 'blacktoken' )
121        )
122 }
123
124 rule InitialMessageState2Place{
125     from
126         h : HALL!InitialMessageState
127     to
128         -- similar to rule NamedMessageState2Place where h.name is
               replaced by 'InitialPlace'
129 }
130
131 rule MessageTransition2Axiom{
132     from
133         h : HALL!MessageTransition
134     to
135         c : COOPN!"COOPNMetamodel::ClassModule::Axiom"(
136            name <- h.name,
137            ownedPre <- sr,
138            ownedPost <- tr,
139            ownedAxiomTheoremSychronisation <- synch,
140            ownedEvent <- evt,
141            ownedCondition <- cond
142        ),
143        sr : COOPN!"COOPNMetamodel::ClassModule::Pre"(
144            ownedPreTerm <- srt
145        ),
146        srt : COOPN!"COOPNMetamodel::ClassModule::Term"(
147            expression <- if(h.transitionsInvMessageState.oclIsTypeOf(
                 HALL!InitialMessageState) ) then
148                    'InitialPlace @'
149                else
150                    h.transitionsInvMessageState.name + ' @'
151                endif
152        ),
153        tr : COOPN!"COOPNMetamodel::ClassModule::Post"(
154            ownedPostTerm <- trt
```

```
155         ),
156         trt : COOPN!"COOPNMetamodel::ClassModule::Term"(
157            expression <- if(h.stateRef.oclIsTypeOf(HALL!
                  InitialMessageState)) then
158                  'InitialPlace @'
159               else
160                  h.stateRef.name + ' @'
161               endif
162         ),
163         synch : COOPN!"COOPNMetamodel::ClassModule::Synchronization"(
164            ownedEventTerms <- Set{syet}
165         ),
166         syet : COOPN!EventTerm(
167            expression <- 'this.fire' + c.name + 'pre'
168         ),
169         evt : COOPN!"COOPNMetamodel::ClassModule::Event"(
170            owenedEventTerm <- Set{evtet}
171         ),
172         evtet : COOPN!"COOPNMetamodel::ClassModule::EventTerm"(
173            expression <- 'fire' + c.name + 'post'
174         ),
175         cond : COOPN!"COOPNMetamodel::ClassModule::Condition"(
176            ownedEquations <- eq
177         ),
178         eq : COOPN!"COOPNMetamodel::ClassModule::Equation"(
179            ownedOperatorEquation <- opeq,
180            ownedEquationTerms <- Set{lterm,rterm}
181         ),
182         lterm : COOPN!"COOPNMetamodel::ClassModule::Term"(
183            expression <- 'this'
184         ),
185         rterm : COOPN!"COOPNMetamodel::ClassModule::Term"(
186            expression <- 'Self'
187         ),
188         opeq : COOPN!"COOPNMetamodel::ClassModule::OperatorEquation"(
189            leftTerm <- lterm,
190            rightTerm <- rterm,
191            operator <- '='
192         )
193  }
```

D.9 mfsm02.atl

```
1  module MFSM02;
2  create coopnModel : COOPN refining preCoopnModel : COOPN, hallModel
       : HALL;
3
4  rule PackageRefactor{
5      from
6          s : COOPN!COOPNPackage
7      to
8          t : COOPN!COOPNPackage(
9              name <- s.name ,
10             ownedModules <- s.ownedModules
11         )
12 }
13
14 rule inherit{
15     from
16         s : COOPN!Inherit
17     to
18         t : COOPN!Inherit(
19         )
20 }
21
22 rule object{
23     from
24         s : COOPN!"COOPNMetamodel::ContextModule::Objects"
25         (s.name = 'FSMObject')
26     to
27         t : COOPN!"COOPNMetamodel::ContextModule::Objects"(
28             name <- s.name
29         )
30 }
31
32 rule sorts{
33     from
34         s : COOPN!"COOPNMetamodel::ADTModule::Sorts"
35     to
36         t : COOPN!"COOPNMetamodel::ADTModule::Sorts"(
37             name <- s.name
38         )
39 }
```

```
40
41  rule contextBodyRefactor{
42      from
43          s : COOPN!"COOPNMetamodel::ContextModule::Body"
44          (HALL!MessageHandler.allInstancesFrom('hallModel')
45                  ->exists(i| let ContextType : String = i.
                        messageHandlerSetInv.oclType().toString()
46                      in ( ContextType.split('!').at(2) + i.
                            messageHandlerSetInv.name ) = s.contextContainsBody
                            .name))
47      to
48          t : COOPN!"COOPNMetamodel::ContextModule::Body"(
49              ownedContextUses <- s.ownedContextUses,
50              ownedBodyUses <- s.ownedBodyUses,
51              ownedObjects <- s.ownedObjects -> union(Set{HALL!
                    MessageHandler.allInstancesFrom('hallModel') -> select (e
                    |
52                  let ContextType : String = e.messageHandlerSetInv.oclType
                        ().toString()
53                  in ( ContextType.split('!').at(2) + e.
                        messageHandlerSetInv.name ) = s.contextContainsBody.
                        name
54                  )})
55          )
56  }
57
58  rule MessageHandler2Object{
59      from
60          h : HALL!MessageHandler
61      to
62          c : COOPN!"COOPNMetamodel::ContextModule::Objects"(
63              name <- h.name + 'MessageHandlerObject' ,
64              objectType <- COOPN!"COOPNMetamodel::ClassModule::ClassType
                    ".allInstancesFrom('preCoopnModel') -> any(e|
65                  let ContextType : String = h.messageHandlerSetInv.oclType
                        ().toString()
66                  in e.name = ( ContextType.split('!').at(2) + h.
                        messageHandlerSetInv.name + h.name + '
                        MessageHandlerType')
67              )
68          )
69  }
70
```

```
71  rule MessageHandlerClassRefactor{
72      from
73          s : COOPN!"COOPNMetamodel::ClassModule::Body"
74          (s.coopnClassContainsBody.name.endsWith('MessageHandler'))
75      to
76          t : COOPN!"COOPNMetamodel::ClassModule::Body"(
77              ownedPlaces <- s.ownedPlaces,
78              ownedInitial <- s.ownedInitial,
79              ownedAxiomTheorems <- s.ownedAxiomTheorems,
80              ownedBodyMethods <- Set{HALL!MessageTransition.
                    allInstancesFrom('hallModel') ->select( e |
81                  if e.transitionsInvMessageState.oclType().toString().
                        endsWith('InitialMessageState')
82                  then s.coopnClassContainsBody.name.endsWith(
83                      e.transitionsInvMessageState.initialMessageStateInv.
                            messageHandlerSetInv.name + e.
                            transitionsInvMessageState.initialMessageStateInv.
                            name + 'MessageHandler') --or t.stateRef.fsm = h
84                  else s.coopnClassContainsBody.name.endsWith(
85                      e.transitionsInvMessageState.messageStateInv.
                            messageHandlerSetInv.name + e.
                            transitionsInvMessageState.messageStateInv.name + '
                            MessageHandler') --or t.stateRef.fsm = h
86                  endif )}
87          )
88
89  }
90
91  rule MessageTransition2Method{
92      from
93          h : HALL!MessageTransition
94      to
95          c : COOPN!"COOPNMetamodel::ClassModule::Methods"(
96              name <- 'fire' + h.name + 'pre'
97          )
98  }
```

D.10 mfsm03.atl

```
1  module MFSM03;
2  create coopnModel : COOPN refining preCoopnModel : COOPN, hallModel
       : HALL;
3
4  rule PackageRefactor{
5      from
6          s : COOPN!COOPNPackage
7      to
8          t : COOPN!COOPNPackage(
9              name <- s.name ,
10             ownedModules <- Set {s.ownedModules}
11         )
12 }
13
14 rule inherit{
15     from
16         s : COOPN!Inherit
17     to
18         t : COOPN!Inherit(
19         )
20 }
21
22 rule object{
23     from
24         s : COOPN!"COOPNMetamodel::ContextModule::Objects"
25     to
26         t : COOPN!"COOPNMetamodel::ContextModule::Objects"(
27             name <- s.name
28         )
29 }
30
31 rule sorts{
32     from
33         s : COOPN!"COOPNMetamodel::ADTModule::Sorts"
34     to
35         t : COOPN!"COOPNMetamodel::ADTModule::Sorts"(
36             name <- s.name
37         )
38 }
39
```

```
40 rule MessageHandlerClassRefactor{
41    from
42       s : COOPN!"COOPNMetamodel::ClassModule::Interface"
43       (s.coopnClassContainsInterface.name.endsWith('MessageHandler'))
44    to
45       t : COOPN!"COOPNMetamodel::ClassModule::Interface"(
46          ownedInterfaceClassTypes <- s.ownedInterfaceClassTypes,
47          ownedInterfaceGates <-
48             Set{HALL!MessageTransition.allInstancesFrom('hallModel')
                  ->select( e |
49             if e.transitionsInvMessageState.oclType().toString().
                  endsWith('InitialMessageState')
50             then s.coopnClassContainsInterface.name.endsWith(
51                e.transitionsInvMessageState.initialMessageStateInv.
                     messageHandlerSetInv.name + e.
                     transitionsInvMessageState.initialMessageStateInv.
                     name + 'MessageHandler') --or t.stateRef.fsm = h
52             else s.coopnClassContainsInterface.name.endsWith(
53                e.transitionsInvMessageState.messageStateInv.
                     messageHandlerSetInv.name + e.
                     transitionsInvMessageState.messageStateInv.name + '
                     MessageHandler') --or t.stateRef.fsm = h
54             endif )}
55       )
56
57 }
58
59 rule MessageTransition2Gates{
60    from
61       h : HALL!MessageTransition
62    to
63       c : COOPN!"COOPNMetamodel::ClassModule::Gates"(
64          name <- 'fire' + h.name + 'post'
65       )
66 }
```

D.11 Data01.atl

```
1  module Data01;
2  create coopnModel : COOPN refining hallModel : HALL, preCoopnModel :
      COOPN, coopnTypes : COOPN;
3
4  rule PackageRefactor{
5      from
6          s : COOPN!COOPNPackage
7          (s.name = 'HALLModelPackage')
8      to
9          t : COOPN!COOPNPackage(
10             name <- s.name ,
11             ownedModules <- Set{s.ownedModules}
12                 ->union(Set{HALL!UserProfile.allInstancesFrom('hallModel
                      ')}->flatten() )
13                 ->union(Set{HALL!SystemComponent.allInstancesFrom('
                      hallModel')}->flatten() )
14                 ->union(Set{HALL!VisualObject.allInstancesFrom('hallModel
                      ')}->flatten() )
15                 ->union(Set{HALL!TaskObject.allInstancesFrom('hallModel')
                      }->flatten() )
16                 ->union(Set{COOPN!COOPNADT.allInstancesFrom('coopnTypes')
                      }->flatten() )
17         )
18 }
19
20 rule inherit{
21     from
22         s : COOPN!Inherit
23     to
24         t : COOPN!Inherit(
25         )
26 }
27
28 rule object{
29     from
30         s : COOPN!"COOPNMetamodel::ContextModule::Objects"
31     to
32         t : COOPN!"COOPNMetamodel::ContextModule::Objects"(
33             name <- s.name
34         )
```

```
35  }
36
37  rule sorts{
38     from
39        s : COOPN!"COOPNMetamodel::ADTModule::Sorts"
40     to
41        t : COOPN!"COOPNMetamodel::ADTModule::Sorts"(
42           name <- s.name
43        )
44  }
45
46  rule userProfile2data{
47     from
48        h : HALL!UserProfile
49     to
50        c : COOPN!"COOPNMetamodel::ClassModule::COOPNClass"(
51           name <- 'UserProfile' + h.name + 'Data',
52           ownedInterface <- int ,
53           ownedBody <- b
54        ),
55        b : COOPN!"COOPNMetamodel::ClassModule::Body"(
56           ownedPlaces <- h.data,
57           ownedVariables <- var
58        ),
59        int : COOPN!"COOPNMetamodel::ClassModule::Interface"(
60           ownedInterfaceClassTypes <- Set{ct}
61        ),
62        ct : COOPN!"COOPNMetamodel::ClassModule::ClassType"(
63           name <- c.name + 'Type',
64           order <- 1
65        ),
66        var : COOPN!"COOPNMetamodel::ClassModule::Variable"(
67           name <- 'this',
68           variableType <- ct
69        )
70  }
71
72  rule systemComponent2data{
73     from
74        h : HALL!SystemComponent
75     to
76        -- similar to rule userProfile2data where UserProfile is
              replaced by SystemComponent
```

```
77  }
78
79  rule taskObject2data{
80      from
81          h : HALL!TaskObject
82      to
83          -- similar to rule userProfile2data where UserProfile is
                replaced by TaskObject
84  }
85
86  rule visualObject2data{
87      from
88          h : HALL!VisualObject
89      to
90          -- similar to rule userProfile2data where UserProfile is
                replaced by VisualObject
91  }
92
93  rule data2place{
94      from
95          h : HALL!Data
96          (not h.dataInvComponent.oclIsUndefined())
97      to
98          c : COOPN!"COOPNMetamodel::ClassModule::Place"(
99              name <- h.name,
100             ownedPlaceTypeElements <- o
101         ),
102         o : COOPN!"COOPNMetamodel::ClassModule::TypesElement"(
103             order <- 1,
104             typeElementType <- COOPN!Sorts.allInstancesFrom('coopnTypes
                    ') -> any( e | e.name = h.type )
105         )
106 }
```

D.12 Data02.atl

```
1  module Data02;
2  create coopnModel : COOPN refining preCoopnModel : COOPN, hallModel :
       HALL;
3
4  rule PackageRefactor{
5      from
6          s : COOPN!COOPNPackage
7      to
8          t : COOPN!COOPNPackage(
9              name <- s.name ,
10             ownedModules <- s.ownedModules
11         )
12 }
13
14 rule inherit{
15     from
16         s : COOPN!Inherit
17     to
18         t : COOPN!Inherit(
19         )
20 }
21
22 rule object{
23     from
24         s : COOPN!"COOPNMetamodel::ContextModule::Objects"
25     to
26         t : COOPN!"COOPNMetamodel::ContextModule::Objects"(
27             name <- s.name
28         )
29 }
30
31 rule sorts{
32     from
33         s : COOPN!"COOPNMetamodel::ADTModule::Sorts"
34     to
35         t : COOPN!"COOPNMetamodel::ADTModule::Sorts"(
36             name <- s.name
37         )
38 }
39
```

```
40
41 rule UserProfilecontextBodyRefactor{
42     from
43         s : COOPN!"COOPNMetamodel::ContextModule::Body"
44         (s.contextContainsBody.name.startsWith('UserProfile'))
45     to
46         t : COOPN!"COOPNMetamodel::ContextModule::Body"(
47             ownedContextUses <- s.ownedContextUses,
48             ownedObjects <- Set{s.ownedObjects}->flatten()
49                 -> union(Set{HALL!UserProfile.allInstancesFrom('hallModel
                        ') -> select(h|
50                         'UserProfile' + h.name = s.contextContainsBody.name
51                     )
52                 }
53             ),
54             ownedBodyUses <- s.ownedBodyUses
55         )
56 }
57
58 rule UserProfileData2Object{
59     from
60         h : HALL!UserProfile
61     to
62         c : COOPN!"COOPNMetamodel::ContextModule::Objects"(
63             name <- 'UserProfile' + h.name + 'DataObject' ,
64             objectType <- COOPN!"COOPNMetamodel::ClassModule::ClassType
                    ".allInstancesFrom('preCoopnModel') -> any(e|
65                 e.name = 'UserProfile' + h.name + 'DataType')
66         )
67 }
68
69 rule SystemComponentBodyRefactor{
70     from
71         s : COOPN!"COOPNMetamodel::ContextModule::Body"
72         (s.contextContainsBody.name.startsWith('SystemComponent'))
73     to
74         -- similar to rule UserProfilecontextBodyRefactor where
               UserProfile is replaced by SystemComponent
75 }
76
77 rule SystemComponentData2Object{
78     from
79         h : HALL!SystemComponent
```

```
80      to
81          -- similar to rule UserProfileData2Object where UserProfile is
                replaced by SystemComponent
82  }
83
84  rule VisualObjectBodyRefactor{
85      from
86          s : COOPN!"COOPNMetamodel::ContextModule::Body"
87          (s.contextContainsBody.name.startsWith('VisualObject'))
88      to
89          -- similar to rule UserProfilecontextBodyRefactor where
                UserProfile is replaced by VisualObject
90  }
91
92  rule VisualObjectData2Object{
93      from
94          h : HALL!VisualObject
95      to
96          -- similar to rule UserProfileData2Object where UserProfile is
                replaced by VisualObject
97  }
98
99  rule TaskObjectBodyRefactor{
100     from
101         s : COOPN!"COOPNMetamodel::ContextModule::Body"
102         (s.contextContainsBody.name.startsWith('TaskObject'))
103     to
104         -- similar to rule UserProfilecontextBodyRefactor where
                UserProfile is replaced by TaskObject
105 }
106
107 rule TaskObjectData2Object{
108     from
109         h : HALL!TaskObject
110     to
111         -- similar to rule UserProfileData2Object where UserProfile is
                replaced by TaskObject
112 }
113
114 rule dataBodyRefactor{
115     from
116         s : COOPN!"COOPNMetamodel::ClassModule::Body"
117         (s.coopnClassContainsBody.name.endsWith('Data'))
```

```
118     to
119         t : COOPN!"COOPNMetamodel::ClassModule::Body"(
120             ownedPlaces <- s.ownedPlaces,
121             ownedVariables <- s.ownedVariables,
122             ownedBodyMethods <- HALL!Data.allInstancesFrom('hallModel')
123                 -> select(h| if not h.dataInvComponent.oclIsUndefined()
124                     then
125                         h.dataInvComponent.oclType().toString().split('!').
                                at(2) +
126                         h.dataInvComponent.name + 'Data'= s.
                                coopnClassContainsBody.name
127                     else
128                         false
129                     endif
130                 )
131         )
132 }
133
134 rule data2method{
135     from
136         h : HALL!Data
137         (not h.dataInvComponent.oclIsUndefined())
138     to
139         c : COOPN!"COOPNMetamodel::ClassModule::Methods"(
140                 name <- 'get' + h.name
141         )
142 }
```

D.13 Data03.atl

```
1  module Data03;
2  create coopnModel : COOPN refining preCoopnModel : COOPN , hallModel
       : HALL;
3
4  rule PackageRefactor{
5      from
6          s : COOPN!COOPNPackage
7      to
8          t : COOPN!COOPNPackage(
9              name <- s.name ,
10             ownedModules <- s.ownedModules
11         )
12 }
13
14 rule inherit{
15     from
16         s : COOPN!Inherit
17     to
18         t : COOPN!Inherit(
19         )
20 }
21
22 rule object{
23     from
24         s : COOPN!"COOPNMetamodel::ContextModule::Objects"
25     to
26         t : COOPN!"COOPNMetamodel::ContextModule::Objects"(
27             name <- s.name
28         )
29 }
30
31 rule sorts{
32     from
33         s : COOPN!"COOPNMetamodel::ADTModule::Sorts"
34     to
35         t : COOPN!"COOPNMetamodel::ADTModule::Sorts"(
36             name <- s.name
37         )
38 }
39
```

```
40 rule dataBodyRefactor{
41     from
42         s : COOPN!"COOPNMetamodel::ClassModule::Body"
43         (s.coopnClassContainsBody.name.endsWith('Data'))
44     to
45         t : COOPN!"COOPNMetamodel::ClassModule::Body"(
46             ownedPlaces <- s.ownedPlaces,
47             ownedVariables <- s.ownedVariables,
48             ownedBodyMethods <- s.ownedBodyMethods -> union( HALL!Data.
                   allInstancesFrom('hallModel')
49                 -> select(h| if not h.dataInvComponent.oclIsUndefined()
50                     then
51                         h.dataInvComponent.oclType().toString().split('!').
                               at(2) +
52                         h.dataInvComponent.name + 'Data'= s.
                               coopnClassContainsBody.name
53                     else
54                         false
55                     endif
56                 )
57             )
58         )
59 }
60
61 rule data2method{
62     from
63         h : HALL!Data
64         (not h.dataInvComponent.oclIsUndefined())
65     to
66         c : COOPN!"COOPNMetamodel::ClassModule::Methods"(
67                 name <- 'set' + h.name
68         )
69 }
```

D.14 Data04.atl

```
1  module Data04;
2  create coopnModel : COOPN refining preCoopnModel : COOPN, hallModel :
       HALL;
3
4  rule PackageRefactor{
5      from
6          s : COOPN!COOPNPackage
7      to
8          t : COOPN!COOPNPackage(
9              name <- s.name ,
10             ownedModules <- s.ownedModules
11         )
12 }
13
14 rule inherit{
15     from
16         s : COOPN!Inherit
17     to
18         t : COOPN!Inherit(
19         )
20 }
21
22 rule object{
23     from
24         s : COOPN!"COOPNMetamodel::ContextModule::Objects"
25     to
26         t : COOPN!"COOPNMetamodel::ContextModule::Objects"(
27             name <- s.name
28         )
29 }
30
31 rule sorts{
32     from
33         s : COOPN!"COOPNMetamodel::ADTModule::Sorts"
34     to
35         t : COOPN!"COOPNMetamodel::ADTModule::Sorts"(
36             name <- s.name
37         )
38 }
39
```

```
40 rule dataBodyRefactor{
41    from
42       s : COOPN!"COOPNMetamodel::ClassModule::Body"
43       (s.coopnClassContainsBody.name.endsWith('Data'))
44    to
45       t : COOPN!"COOPNMetamodel::ClassModule::Body"(
46          ownedPlaces <- s.ownedPlaces,
47          ownedVariables <- s.ownedVariables,
48          ownedBodyMethods <- s.ownedBodyMethods,
49          ownedAxiomTheorems <- HALL!Data.allInstancesFrom('hallModel
              ')
50             -> select(h| if not h.dataInvComponent.oclIsUndefined()
51                then
52                   h.dataInvComponent.oclType().toString().split('!').
                        at(2)
53                   + h.dataInvComponent.name + 'Data'= s.
                        coopnClassContainsBody.name
54                else
55                   false
56                endif
57             )
58       )
59 }
60
61
62 rule method2axiom{
63    from
64       h : HALL!Data
65       (not h.dataInvComponent.oclIsUndefined())
66    to
67       c : COOPN!"COOPNMetamodel::ClassModule::Axiom"(
68          name <- 'set' + h.name + 'Axiom',
69          ownedPre <- sr,
70          ownedPost <- tr,
71          ownedAxiomTheoremSychronisation <- synch,
72          ownedEvent <- evt,
73          ownedCondition <- cond
74       ),
75       sr : COOPN!"COOPNMetamodel::ClassModule::Pre"(
76          ownedPreTerm <- srt
77       ),
78       srt : COOPN!"COOPNMetamodel::ClassModule::Term"(
79          expression <- h.name + ' @'
```

```
80          ),
81          tr : COOPN!"COOPNMetamodel::ClassModule::Post"(
82              ownedPostTerm <- trt
83          ),
84          trt : COOPN!"COOPNMetamodel::ClassModule::Term"(
85              expression <- h.name + ' @'
86          ),
87          synch : COOPN!"COOPNMetamodel::ClassModule::Synchronization"(
88              ownedEventTerms <- Set{syet}
89          ),
90          syet : COOPN!EventTerm(
91              expression <- 'this.fire' + c.name + 'post'
92          ),
93          evt : COOPN!"COOPNMetamodel::ClassModule::Event"(
94              owenedEventTerm <- Set{evtet}
95          ),
96          evtet : COOPN!"COOPNMetamodel::ClassModule::EventTerm"(
97              expression <- 'fire' + c.name + 'pre'
98          ),
99          cond : COOPN!"COOPNMetamodel::ClassModule::Condition"(
100             ownedEquations <- eq
101         ),
102         eq : COOPN!"COOPNMetamodel::ClassModule::Equation"(
103             ownedOperatorEquation <- opeq,
104             ownedEquationTerms <- Set{lterm,rterm}
105         ),
106         lterm : COOPN!"COOPNMetamodel::ClassModule::Term"(
107             expression <- 'this'
108         ),
109         rterm : COOPN!"COOPNMetamodel::ClassModule::Term"(
110             expression <- 'Self'
111         ),
112         opeq : COOPN!"COOPNMetamodel::ClassModule::OperatorEquation"(
113             leftTerm <- lterm,
114             rightTerm <- rterm,
115             operator <- '='
116         )
117 }
```

D.15 Data05.atl

```
1  module Data05;
2  create coopnModel : COOPN refining preCoopnModel : COOPN, hallModel :
       HALL;
3
4  rule PackageRefactor{
5      from
6          s : COOPN!COOPNPackage
7      to
8          t : COOPN!COOPNPackage(
9              name <- s.name ,
10             ownedModules <- s.ownedModules
11         )
12 }
13
14 rule inherit{
15     from
16         s : COOPN!Inherit
17     to
18         t : COOPN!Inherit(
19         )
20 }
21
22 rule object{
23     from
24         s : COOPN!"COOPNMetamodel::ContextModule::Objects"
25     to
26         t : COOPN!"COOPNMetamodel::ContextModule::Objects"(
27             name <- s.name
28         )
29 }
30
31 rule sorts{
32     from
33         s : COOPN!"COOPNMetamodel::ADTModule::Sorts"
34     to
35         t : COOPN!"COOPNMetamodel::ADTModule::Sorts"(
36             name <- s.name
37         )
38 }
39
```

```
40  rule dataBodyRefactor{
41      from
42          s : COOPN!"COOPNMetamodel::ClassModule::Body"
43          (s.coopnClassContainsBody.name.endsWith('Data'))
44      to
45          t : COOPN!"COOPNMetamodel::ClassModule::Body"(
46              ownedPlaces <- s.ownedPlaces,
47              ownedVariables <- s.ownedVariables,
48              ownedBodyMethods <- s.ownedBodyMethods,
49              ownedAxiomTheorems <- s.ownedAxiomTheorems -> union( HALL!
                    Data.allInstancesFrom('hallModel')
50                  -> select(h| if not h.dataInvComponent.oclIsUndefined()
51                      then
52                          h.dataInvComponent.oclType().toString().split('!').
                                at(2)
53                          + h.dataInvComponent.name + 'Data'= s.
                                coopnClassContainsBody.name
54                      else
55                          false
56                      endif
57                  )
58              )
59          )
60  }
61
62
63  rule method2axiom{
64      from
65          h : HALL!Data
66          (not h.dataInvComponent.oclIsUndefined())
67      to
68          c : COOPN!"COOPNMetamodel::ClassModule::Axiom"(
69              name <- 'get' + h.name + 'Axiom',
70              ownedPre <- sr,
71              ownedPost <- tr,
72              ownedAxiomTheoremSychronisation <- synch,
73              ownedEvent <- evt,
74              ownedCondition <- cond
75          ),
76          sr : COOPN!"COOPNMetamodel::ClassModule::Pre"(
77              ownedPreTerm <- srt
78          ),
79          srt : COOPN!"COOPNMetamodel::ClassModule::Term"(
```

```
80            expression <- h.name + ' @'
81         ),
82         tr : COOPN!"COOPNMetamodel::ClassModule::Post"(
83            ownedPostTerm <- trt
84         ),
85         trt : COOPN!"COOPNMetamodel::ClassModule::Term"(
86            expression <- h.name + ' @'
87         ),
88         synch : COOPN!"COOPNMetamodel::ClassModule::Synchronization"(
89            ownedEventTerms <- Set{syet}
90         ),
91         syet : COOPN!EventTerm(
92            expression <- 'this.fire' + c.name + 'post'
93         ),
94         evt : COOPN!"COOPNMetamodel::ClassModule::Event"(
95            owenedEventTerm <- Set{evtet}
96         ),
97         evtet : COOPN!"COOPNMetamodel::ClassModule::EventTerm"(
98            expression <- 'fire' + c.name + 'pre'
99         ),
100        cond : COOPN!"COOPNMetamodel::ClassModule::Condition"(
101           ownedEquations <- eq
102        ),
103        eq : COOPN!"COOPNMetamodel::ClassModule::Equation"(
104           ownedOperatorEquation <- opeq,
105           ownedEquationTerms <- Set{lterm,rterm}
106        ),
107        lterm : COOPN!"COOPNMetamodel::ClassModule::Term"(
108           expression <- 'this'
109        ),
110        rterm : COOPN!"COOPNMetamodel::ClassModule::Term"(
111           expression <- 'Self'
112        ),
113        opeq : COOPN!"COOPNMetamodel::ClassModule::OperatorEquation"(
114           leftTerm <- lterm,
115           rightTerm <- rterm,
116           operator <- '='
117        )
118 }
```

D.16 Router01.atl

```
1  module Router01; -- Module Template
2  create coopnModel : COOPN refining hallModel : HALL, preCoopnModel :
       COOPN;
3
4  rule PackageRefactor{
5      from
6          s : COOPN!COOPNPackage
7      to
8          t : COOPN!COOPNPackage(
9              name <- s.name ,
10             ownedModules <- Set{s.ownedModules}
11                 ->union(Set{HALL!UserProfile.allInstancesFrom('hallModel
                       ')}->flatten() )
12                 ->union(Set{HALL!SystemComponent.allInstancesFrom('
                       hallModel')}->flatten() )
13                 ->union(Set{HALL!VisualObject.allInstancesFrom('hallModel
                       ')}->flatten() )
14                 ->union(Set{HALL!TaskObject.allInstancesFrom('hallModel')
                       }->flatten() )
15         )
16 }
17
18 rule inherit{
19     from
20         s : COOPN!Inherit
21     to
22         t : COOPN!Inherit(
23         )
24 }
25
26 rule object{
27     from
28         s : COOPN!"COOPNMetamodel::ContextModule::Objects"
29     to
30         t : COOPN!"COOPNMetamodel::ContextModule::Objects"(
31             name <- s.name
32         )
33 }
34
35 rule sorts{
```

```
36     from
37         s : COOPN!"COOPNMetamodel::ADTModule::Sorts"
38     to
39         t : COOPN!"COOPNMetamodel::ADTModule::Sorts"(
40             name <- s.name
41         )
42 }
43
44 rule userProfile2Router{
45     from
46         h : HALL!UserProfile
47     to
48         c : COOPN!"COOPNMetamodel::ClassModule::COOPNClass"(
49             name <- 'UserProfile' + h.name + 'Router',
50             ownedInterface <- int ,
51             ownedBody <- b
52         ),
53         b : COOPN!"COOPNMetamodel::ClassModule::Body"(
54             ownedPlaces <- Set{re},
55             ownedVariables <- Set{var, mname, mparam, mdirection}
56         ),
57         int : COOPN!"COOPNMetamodel::ClassModule::Interface"(
58             ownedInterfaceClassTypes <- Set{ct},
59             ownedInterfaceGates <- Set{gg}
60                 ->union( Set{HALL!MessageTransition.allInstancesFrom('
                       hallModel') ->select( e |
61                     if e.transitionsInvMessageState.oclType().toString().
                           endsWith('InitialMessageState')
62                     then c.name.endsWith(
63                         e.transitionsInvMessageState.initialMessageStateInv
                               .messageHandlerSetInv.name + 'Router')
64                     else c.name.endsWith(
65                         e.transitionsInvMessageState.messageStateInv.
                               messageHandlerSetInv.name + 'Router')
66                     endif )} )
67                 ->union( Set{HALL!Transition.allInstancesFrom('hallModel
                       ')->select( e |
68                     c.name.endsWith(e.source.fsm.FSMInv.name + 'Router') )
                           }
69                 ),
70             ownedInterfaceMethods <- Set{gm}
71         ),
72         ct : COOPN!"COOPNMetamodel::ClassModule::ClassType"(
```

```
73             name <- c.name + 'Type',
74             order <- 1
75         ),
76         var : COOPN!"COOPNMetamodel::ClassModule::Variable"(
77             name <- 'this',
78             variableType <- ct
79         ),
80         mname : COOPN!"COOPNMetamodel::ClassModule::Variable"(
81             name <- 'mname',
82             variableType <- COOPN!Sorts.allInstancesFrom('preCoopnModel
                   ') -> any( e | e.name = 'string' )
83         ),
84         mparam : COOPN!"COOPNMetamodel::ClassModule::Variable"(
85             name <- 'mparam',
86             variableType <- COOPN!Sorts.allInstancesFrom('preCoopnModel
                   ') -> any( e | e.name = 'natural' )
87         ),
88         mdirection : COOPN!"COOPNMetamodel::ClassModule::Variable"(
89             name <- 'mdirection',
90             variableType <- COOPN!Sorts.allInstancesFrom('preCoopnModel
                   ') -> any( e | e.name = 'boolean' )
91         ),
92         re : COOPN!"COOPNMetamodel::ClassModule::Place"(
93                 name <- c.name + 'Enabled',
94                 ownedPlaceTypeElements <- reo
95         ),
96         reo : COOPN!"COOPNMetamodel::ClassModule::TypesElement"(
97             order <- 1,
98             typeElementType <- COOPN!Sorts.allInstancesFrom('
                   preCoopnModel') -> any( e | e.name = 'blacktoken' )
99         ),
100        gg : COOPN!"COOPNMetamodel::ClassModule::Gates"(
101                name <- 'fire' + h.name + 'RouterPost'
102        ),
103        gm : COOPN!"COOPNMetamodel::ClassModule::Methods"(
104                name <- 'fire' + h.name + 'RouterPre'
105        )
106 }
107
108 rule systemComponent2Router{
109     from
110         h : HALL!SystemComponent
111     to
```

```
112         -- similar to rule userProfile2Router where UserProfile is
               replaced by SystemComponent
113  }
114
115  rule taskObject2Router{
116      from
117          h : HALL!TaskObject
118      to
119          -- similar to rule userProfile2Router where UserProfile is
               replaced by TaskObject
120  }
121
122  rule visualObject2Router{
123      from
124          h : HALL!VisualObject
125      to
126          -- similar to rule userProfile2Router where UserProfile is
               replaced by VisualObject
127  }
128
129  rule MessageTransition2Gate{
130      from
131          h : HALL!MessageTransition
132      to
133          c : COOPN!"COOPNMetamodel::ClassModule::Gates"(
134              name <- if h.transitionsInvMessageState.oclType().toString()
                     .endsWith('InitialMessageState')
135                  then 'fire' + h.transitionsInvMessageState.
                         initialMessageStateInv.messageHandlerSetInv.name +
                         'Router'+ h.name + 'post'
136                  else 'fire' + h.transitionsInvMessageState.
                         messageStateInv.messageHandlerSetInv.name + 'Router
                         ' + h.name + 'post'
137                  endif
138          )
139  }
140
141  rule Transition2Gate{
142      from
143          h : HALL!Transition
144      to
145          c : COOPN!"COOPNMetamodel::ClassModule::Gates"(
```

```
146             name <- 'fire' + h.source.fsm.FSMInv.name + 'Router' + h.
                    name + 'post'
147         )
148 }
```

D.17 Router02.atl

```
1  module Router02;
2  create coopnModel : COOPN refining hallModel : HALL, preCoopnModel :
      COOPN;
3
4  rule PackageRefactor{
5      from
6          s : COOPN!COOPNPackage
7      to
8          t : COOPN!COOPNPackage(
9              name <- s.name ,
10             ownedModules <- s.ownedModules
11         )
12 }
13
14 rule inherit{
15     from
16         s : COOPN!Inherit
17     to
18         t : COOPN!Inherit(
19         )
20 }
21
22 rule object{
23     from
24         s : COOPN!"COOPNMetamodel::ContextModule::Objects"
25     to
26         t : COOPN!"COOPNMetamodel::ContextModule::Objects"(
27             name <- s.name
28         )
29 }
30
31 rule sorts{
32     from
33         s : COOPN!"COOPNMetamodel::ADTModule::Sorts"
34     to
35         t : COOPN!"COOPNMetamodel::ADTModule::Sorts"(
36             name <- s.name
37         )
38 }
39
```

```
40 rule RouterClassRefactor{
41     from
42         s : COOPN!"COOPNMetamodel::ClassModule::Interface"
43         (s.coopnClassContainsInterface.name.endsWith('Router'))
44     to
45         t : COOPN!"COOPNMetamodel::ClassModule::Interface"(
46             ownedInterfaceClassTypes <- s.ownedInterfaceClassTypes,
47             ownedInterfaceGates <- s.ownedInterfaceGates,
48             ownedInterfaceMethods <- Set{s.ownedInterfaceMethods}
49                 ->union( Set{HALL!MessageTransition.allInstancesFrom('
                        hallModel') ->select( e |
50                     if e.transitionsInvMessageState.oclType().toString().
                            endsWith('InitialMessageState')
51                     then s.coopnClassContainsInterface.name.endsWith(
52                         e.transitionsInvMessageState.initialMessageStateInv
                                .messageHandlerSetInv.name + 'Router')
53                     else s.coopnClassContainsInterface.name.endsWith(
54                         e.transitionsInvMessageState.messageStateInv.
                                messageHandlerSetInv.name + 'Router')
55                     endif )} )
56                 ->union( Set{HALL!Transition.allInstancesFrom('hallModel
                        ')->select( e |
57                     s.coopnClassContainsInterface.name.endsWith(e.source.
                            fsm.FSMInv.name + 'Router') )}
58                 )
59         )
60 }
61
62 rule MessageTransition2Methods{
63     from
64         h : HALL!MessageTransition
65     to
66         c : COOPN!"COOPNMetamodel::ClassModule::Methods"(
67             name <- if h.transitionsInvMessageState.oclType().toString()
                    .endsWith('InitialMessageState')
68                     then h.transitionsInvMessageState.
                            initialMessageStateInv.messageHandlerSetInv.name +
                            'Router'+ h.name + 'pre'
69                     else h.transitionsInvMessageState.messageStateInv.
                            messageHandlerSetInv.name + 'Router' + h.name + '
                            pre'
70                     endif
71             )
```

```
72  }
73
74  rule Transition2Methods{
75      from
76          h : HALL!Transition
77      to
78          c : COOPN!"COOPNMetamodel::ClassModule::Methods"(
79              name <- 'fire' + h.source.fsm.FSMInv.name + 'Router' + h.
                    name + 'pre'
80          )
81  }
```

D.18 Router03.atl

```
1  module Router03;
2  create coopnModel : COOPN refining hallModel : HALL, preCoopnModel :
      COOPN;
3
4  rule PackageRefactor{
5      from
6          s : COOPN!COOPNPackage
7      to
8          t : COOPN!COOPNPackage(
9              name <- s.name ,
10             ownedModules <- s.ownedModules
11         )
12 }
13
14 rule inherit{
15     from
16         s : COOPN!Inherit
17     to
18         t : COOPN!Inherit(
19         )
20 }
21
22 rule object{
23     from
24         s : COOPN!"COOPNMetamodel::ContextModule::Objects"
25     to
26         t : COOPN!"COOPNMetamodel::ContextModule::Objects"(
27             name <- s.name
28         )
29 }
30
31 rule sorts{
32     from
33         s : COOPN!"COOPNMetamodel::ADTModule::Sorts"
34     to
35         t : COOPN!"COOPNMetamodel::ADTModule::Sorts"(
36             name <- s.name
37         )
38 }
39
```

```
40 rule RouterClassRefactor{
41     from
42         s : COOPN!"COOPNMetamodel::ClassModule::Body"
43         (s.coopnClassContainsBody.name.endsWith('Router'))
44     to
45         t : COOPN!"COOPNMetamodel::ClassModule::Body"(
46             ownedPlaces <- s.ownedPlaces,
47             ownedInitial <- s.ownedInitial,
48             ownedVariables <- s.ownedVariables,
49             ownedAxiomTheorems <- Set{HALL!MessageTransition.
                   allInstancesFrom('hallModel') ->select( e |
50                     if e.transitionsInvMessageState.oclType().toString().
                           endsWith('InitialMessageState')
51                     then s.coopnClassContainsBody.name.endsWith(
52                         e.transitionsInvMessageState.initialMessageStateInv
                               .messageHandlerSetInv.name + 'Router')
53                     else s.coopnClassContainsBody.name.endsWith(
54                         e.transitionsInvMessageState.messageStateInv.
                               messageHandlerSetInv.name + 'Router')
55                     endif )}
56                 ->union( Set{HALL!Transition.allInstancesFrom('hallModel
                       ')->select( e |
57                     s.coopnClassContainsBody.name.endsWith(e.source.fsm.
                           FSMInv.name + 'Router') )}
58                 )
59         )
60 
61 }
62 
63 rule MessageTransition2Axiom{
64     from
65         h : HALL!MessageTransition
66     to
67         c : COOPN!"COOPNMetamodel::ClassModule::Axiom"(
68             name <- h.name,
69             ownedPre <- sr,
70             ownedPost <- tr,
71             ownedAxiomTheoremSychronisation <- synch,
72             ownedEvent <- evt,
73             ownedCondition <- cond
74         ),
75         sr : COOPN!"COOPNMetamodel::ClassModule::Pre"(
76             ownedPreTerm <- srt
```

```
77          ),
78          srt : COOPN!"COOPNMetamodel::ClassModule::Term"(
79              expression <- if(h.transitionsInvMessageState.oclType().
                    toString().endsWith('InitialMessageState')) then
80                          h.transitionsInvMessageState.
                                initialMessageStateInv.messageHandlerSetInv.
                                oclType().toString().split('!').at(2)
81                              + h.transitionsInvMessageState.
                                    initialMessageStateInv.
                                    messageHandlerSetInv.name + 'RouterEnabled
                                     @'
82                      else
83                          h.transitionsInvMessageState.messageStateInv.
                                messageHandlerSetInv.oclType().toString().
                                split('!').at(2)
84                              + h.transitionsInvMessageState.
                                    messageStateInv.messageHandlerSetInv.name
                                    + 'RouterEnabled @'
85                      endif
86          ),
87          tr : COOPN!"COOPNMetamodel::ClassModule::Post"(
88              ownedPostTerm <- trt
89          ),
90          trt : COOPN!"COOPNMetamodel::ClassModule::Term"(
91              expression <- if(h.transitionsInvMessageState.oclType().
                    toString().endsWith('InitialMessageState')) then
92                          h.transitionsInvMessageState.
                                initialMessageStateInv.messageHandlerSetInv.
                                oclType().toString().split('!').at(2)
93                              + h.transitionsInvMessageState.
                                    initialMessageStateInv.
                                    messageHandlerSetInv.name + 'RouterEnabled
                                     @'
94                      else
95                          h.transitionsInvMessageState.messageStateInv.
                                messageHandlerSetInv.oclType().toString().
                                split('!').at(2)
96                              + h.transitionsInvMessageState.
                                    messageStateInv.messageHandlerSetInv.name
                                    + 'RouterEnabled @'
97                      endif
98          ),
99          synch : COOPN!"COOPNMetamodel::ClassModule::Synchronization"(
```

```
100            ownedEventTerms <- Set{syet}
101         ),
102         syet : COOPN!EventTerm(
103            expression <- if h.transitionsInvMessageState.oclType().
                  toString().endsWith('InitialMessageState')
104                  then 'this.fire' + h.transitionsInvMessageState.
                        initialMessageStateInv.messageHandlerSetInv.name +
                        'Router'+ h.name + 'post mname mparam mdirection'
105                  else 'this.fire' + h.transitionsInvMessageState.
                        messageStateInv.messageHandlerSetInv.name + 'Router
                        ' + h.name + 'post mname mparam mdirection'
106                  endif
107         ),
108         evt : COOPN!"COOPNMetamodel::ClassModule::Event"(
109            owenedEventTerm <- Set{evtet}
110         ),
111         evtet : COOPN!"COOPNMetamodel::ClassModule::EventTerm"(
112            expression <- if h.transitionsInvMessageState.oclType().
                  toString().endsWith('InitialMessageState')
113                  then 'fire' + h.transitionsInvMessageState.
                        initialMessageStateInv.messageHandlerSetInv.name +
                        'RouterPre mname mparam mdirection'
114                  else 'fire' + h.transitionsInvMessageState.
                        messageStateInv.messageHandlerSetInv.name + '
                        RouterPre mname mparam mdirection'
115                  endif
116         ),
117         cond : COOPN!"COOPNMetamodel::ClassModule::Condition"(
118            ownedEquations <- Set{eq,meq}
119         ),
120         eq : COOPN!"COOPNMetamodel::ClassModule::Equation"(
121            ownedOperatorEquation <- opeq,
122            ownedEquationTerms <- Set{lterm,rterm}
123         ),
124         lterm : COOPN!"COOPNMetamodel::ClassModule::Term"(
125            expression <- 'this'
126         ),
127         rterm : COOPN!"COOPNMetamodel::ClassModule::Term"(
128            expression <- 'Self'
129         ),
130         opeq : COOPN!"COOPNMetamodel::ClassModule::OperatorEquation"(
131            leftTerm <- lterm,
132            rightTerm <- rterm,
```

```
133             operator <- '='
134         ),
135         meq : COOPN!"COOPNMetamodel::ClassModule::Equation"(
136             ownedOperatorEquation <- mopeq,
137             ownedEquationTerms <- Set{mlterm,mrterm}
138         ),
139         mlterm : COOPN!"COOPNMetamodel::ClassModule::Term"(
140             expression <- 'mname'
141         ),
142         mrterm : COOPN!"COOPNMetamodel::ClassModule::Term"(
143             expression <- h.name
144         ),
145         mopeq : COOPN!"COOPNMetamodel::ClassModule::OperatorEquation"(
146             leftTerm <- mlterm,
147             rightTerm <- mrterm,
148             operator <- '='
149         )
150 }
151
152 rule Transition2Axiom{
153     from
154         h : HALL!Transition
155     to
156         c : COOPN!"COOPNMetamodel::ClassModule::Axiom"(
157             name <- h.name,
158             ownedPre <- sr,
159             ownedPost <- tr,
160             ownedAxiomTheoremSychronisation <- synch,
161             ownedEvent <- evt,
162             ownedCondition <- cond
163         ),
164         sr : COOPN!"COOPNMetamodel::ClassModule::Pre"(
165             ownedPreTerm <- srt
166         ),
167         srt : COOPN!"COOPNMetamodel::ClassModule::Term"(
168             expression <- h.source.fsm.FSMInv.oclType().toString().
                    split('!').at(2)
169                         + h.source.fsm.FSMInv.name + 'RouterEnabled @'
170         ),
171         tr : COOPN!"COOPNMetamodel::ClassModule::Post"(
172             ownedPostTerm <- trt
173         ),
174         trt : COOPN!"COOPNMetamodel::ClassModule::Term"(
```

```
175             expression <- h.source.fsm.FSMInv.oclType().toString().split
                   ('!').at(2)
176                         + h.source.fsm.FSMInv.name + 'RouterEnabled @'
177         ),
178         synch : COOPN!"COOPNMetamodel::ClassModule::Synchronization"(
179             ownedEventTerms <- Set{syet}
180         ),
181         syet : COOPN!EventTerm(
182             expression <- 'this.fire ' + h.source.fsm.FSMInv.name + '
                   Router' + h.name + 'post mname mparam mdirection'
183         ),
184         evt : COOPN!"COOPNMetamodel::ClassModule::Event"(
185             owenedEventTerm <- Set{evtet}
186         ),
187         evtet : COOPN!"COOPNMetamodel::ClassModule::EventTerm"(
188             expression <- 'fire ' + h.source.fsm.FSMInv.name + 'RouterPre
                    mname mparam mdirection'
189         ),
190         cond : COOPN!"COOPNMetamodel::ClassModule::Condition"(
191             ownedEquations <- Set{eq,meq}
192         ),
193         eq : COOPN!"COOPNMetamodel::ClassModule::Equation"(
194             ownedOperatorEquation <- opeq,
195             ownedEquationTerms <- Set{lterm,rterm}
196         ),
197         lterm : COOPN!"COOPNMetamodel::ClassModule::Term"(
198             expression <- 'this'
199         ),
200         rterm : COOPN!"COOPNMetamodel::ClassModule::Term"(
201             expression <- 'Self'
202         ),
203         opeq : COOPN!"COOPNMetamodel::ClassModule::OperatorEquation"(
204             leftTerm <- lterm,
205             rightTerm <- rterm,
206             operator <- '='
207         ),
208         meq : COOPN!"COOPNMetamodel::ClassModule::Equation"(
209             ownedOperatorEquation <- mopeq,
210             ownedEquationTerms <- Set{mlterm,mrterm}
211         ),
212         mlterm : COOPN!"COOPNMetamodel::ClassModule::Term"(
213             expression <- 'mname'
214         ),
```

```
215         mrterm  : COOPN!"COOPNMetamodel::ClassModule::Term"(
216             expression <- h.name
217         ),
218         mopeq  : COOPN!"COOPNMetamodel::ClassModule::OperatorEquation"(
219             leftTerm <- mlterm,
220             rightTerm <- mrterm,
221             operator <- '='
222         )
223 }
```

D.19 Router04.atl

```
1  module Router04;
2  create coopnModel : COOPN refining hallModel : HALL, preCoopnModel :
      COOPN;
3
4  rule PackageRefactor{
5      from
6          s : COOPN!COOPNPackage
7      to
8          t : COOPN!COOPNPackage(
9              name <- s.name ,
10             ownedModules <- s.ownedModules
11         )
12 }
13
14 rule inherit{
15     from
16         s : COOPN!Inherit
17     to
18         t : COOPN!Inherit(
19         )
20 }
21
22 rule object{
23     from
24         s : COOPN!"COOPNMetamodel::ContextModule::Objects"
25     to
26         t : COOPN!"COOPNMetamodel::ContextModule::Objects"(
27             name <- s.name
28         )
29 }
30
31 rule sorts{
32     from
33         s : COOPN!"COOPNMetamodel::ADTModule::Sorts"
34     to
35         t : COOPN!"COOPNMetamodel::ADTModule::Sorts"(
36             name <- s.name
37         )
38 }
39
```

```
40 rule RouterClassRefactor{
41     from
42         s : COOPN!"COOPNMetamodel::ClassModule::Body"
43         (s.coopnClassContainsBody.name.endsWith('Router'))
44     to
45         t : COOPN!"COOPNMetamodel::ClassModule::Body"(
46             ownedPlaces <- s.ownedPlaces,
47             ownedInitial <- s.ownedInitial,
48             ownedVariables <- s.ownedVariables,
49             ownedAxiomTheorems <- Set{s.ownedAxiomTheorems}
50                 ->union(Set{HALL!MessageTransition.allInstancesFrom('
                       hallModel')->select( e |
51                     if e.transitionsInvMessageState.oclType().toString().
                           endsWith('InitialMessageState')
52                     then s.coopnClassContainsBody.name.endsWith(
53                         e.transitionsInvMessageState.initialMessageStateInv
                               .messageHandlerSetInv.name + 'Router')
54                     else s.coopnClassContainsBody.name.endsWith(
55                         e.transitionsInvMessageState.messageStateInv.
                               messageHandlerSetInv.name + 'Router')
56                     endif )} )
57                 ->union( Set{HALL!Transition.allInstancesFrom('hallModel
                       ')->select( e |
58                     s.coopnClassContainsBody.name.endsWith(e.source.fsm.
                           FSMInv.name + 'Router') )}
59                 )
60         )
61 }
62
63 rule MessageTransition2Axiom{
64     from
65         h : HALL!MessageTransition
66     to
67         c : COOPN!"COOPNMetamodel::ClassModule::Axiom"(
68             name <- h.name,
69             ownedPre <- sr,
70             ownedPost <- tr,
71             ownedAxiomTheoremSychronisation <- synch,
72             ownedEvent <- evt,
73             ownedCondition <- cond
74         ),
75         sr : COOPN!"COOPNMetamodel::ClassModule::Pre"(
76             ownedPreTerm <- srt
```

```
77        ),
78        srt : COOPN!"COOPNMetamodel::ClassModule::Term"(
79           expression <- if(h.transitionsInvMessageState.oclType().
                toString().endsWith('InitialMessageState')) then
80                    h.transitionsInvMessageState.
                          initialMessageStateInv.messageHandlerSetInv.
                          oclType().toString().split('!').at(2)
81                        + h.transitionsInvMessageState.
                             initialMessageStateInv.
                             messageHandlerSetInv.name + 'RouterEnabled
                              @'
82                 else
83                    h.transitionsInvMessageState.messageStateInv.
                          messageHandlerSetInv.oclType().toString().
                          split('!').at(2)
84                        + h.transitionsInvMessageState.
                             messageStateInv.messageHandlerSetInv.name
                             + 'RouterEnabled @'
85                 endif
86        ),
87        tr : COOPN!"COOPNMetamodel::ClassModule::Post"(
88           ownedPostTerm <- trt
89        ),
90        trt : COOPN!"COOPNMetamodel::ClassModule::Term"(
91           expression <- if(h.transitionsInvMessageState.oclType().
                toString().endsWith('InitialMessageState')) then
92                    h.transitionsInvMessageState.
                          initialMessageStateInv.messageHandlerSetInv.
                          oclType().toString().split('!').at(2)
93                        + h.transitionsInvMessageState.
                             initialMessageStateInv.
                             messageHandlerSetInv.name + 'RouterEnabled
                              @'
94                 else
95                    h.transitionsInvMessageState.messageStateInv.
                          messageHandlerSetInv.oclType().toString().
                          split('!').at(2)
96                        + h.transitionsInvMessageState.
                             messageStateInv.messageHandlerSetInv.name
                             + 'RouterEnabled @'
97                 endif
98        ),
99        synch : COOPN!"COOPNMetamodel::ClassModule::Synchronization"(
```

```
100             ownedEventTerms <- Set{syet}
101         ),
102         syet : COOPN!EventTerm(
103             expression <- if h.transitionsInvMessageState.oclType().
                    toString().endsWith('InitialMessageState')
104                     then 'this.fire' + h.transitionsInvMessageState.
                            initialMessageStateInv.messageHandlerSetInv.name +
                            'RouterPost mname mparam mdirection'
105                     else 'this.fire' + h.transitionsInvMessageState.
                            messageStateInv.messageHandlerSetInv.name + '
                            RouterPost mname mparam mdirection'
106                     endif
107         ),
108         evt : COOPN!"COOPNMetamodel::ClassModule::Event"(
109             owenedEventTerm <- Set{evtet}
110         ),
111         evtet : COOPN!"COOPNMetamodel::ClassModule::EventTerm"(
112             expression <- if h.transitionsInvMessageState.oclType().
                    toString().endsWith('InitialMessageState')
113                     then 'fire' + h.transitionsInvMessageState.
                            initialMessageStateInv.messageHandlerSetInv.name +
                            'Router'+ h.name + 'pre mname mparam mdirection'
114                     else 'fire' + h.transitionsInvMessageState.
                            messageStateInv.messageHandlerSetInv.name + 'Router
                            ' + h.name + 'pre mname mparam mdirection'
115                     endif
116         ),
117         cond : COOPN!"COOPNMetamodel::ClassModule::Condition"(
118             ownedEquations <- Set{eq,meq}
119         ),
120         eq : COOPN!"COOPNMetamodel::ClassModule::Equation"(
121             ownedOperatorEquation <- opeq,
122             ownedEquationTerms <- Set{lterm,rterm}
123         ),
124         lterm : COOPN!"COOPNMetamodel::ClassModule::Term"(
125             expression <- 'this'
126         ),
127         rterm : COOPN!"COOPNMetamodel::ClassModule::Term"(
128             expression <- 'Self'
129         ),
130         opeq : COOPN!"COOPNMetamodel::ClassModule::OperatorEquation"(
131             leftTerm <- lterm,
132             rightTerm <- rterm,
```

```
133             operator <- '='
134         ),
135         meq : COOPN!"COOPNMetamodel::ClassModule::Equation"(
136             ownedOperatorEquation <- mopeq,
137             ownedEquationTerms <- Set{mlterm,mrterm}
138         ),
139         mlterm : COOPN!"COOPNMetamodel::ClassModule::Term"(
140             expression <- 'mname'
141         ),
142         mrterm : COOPN!"COOPNMetamodel::ClassModule::Term"(
143             expression <- h.name
144         ),
145         mopeq : COOPN!"COOPNMetamodel::ClassModule::OperatorEquation"(
146             leftTerm <- mlterm,
147             rightTerm <- mrterm,
148             operator <- '='
149         )
150 }
151
152 rule Transition2Axiom{
153     from
154         h : HALL!Transition
155     to
156         c : COOPN!"COOPNMetamodel::ClassModule::Axiom"(
157             name <- h.name,
158             ownedPre <- sr,
159             ownedPost <- tr,
160             ownedAxiomTheoremSychronisation <- synch,
161             ownedEvent <- evt,
162             ownedCondition <- cond
163         ),
164         sr : COOPN!"COOPNMetamodel::ClassModule::Pre"(
165             ownedPreTerm <- srt
166         ),
167         srt : COOPN!"COOPNMetamodel::ClassModule::Term"(
168             expression <- h.source.fsm.FSMInv.oclType().toString().split
                    ('!').at(2)
169                         + h.source.fsm.FSMInv.name + 'RouterEnabled @'
170         ),
171         tr : COOPN!"COOPNMetamodel::ClassModule::Post"(
172             ownedPostTerm <- trt
173         ),
174         trt : COOPN!"COOPNMetamodel::ClassModule::Term"(
```

```
175             expression <- h.source.fsm.FSMInv.oclType().toString().split
                    ('!').at(2)
176                         + h.source.fsm.FSMInv.name + 'RouterEnabled @'
177         ),
178         synch : COOPN!"COOPNMetamodel::ClassModule::Synchronization"(
179             ownedEventTerms <- Set{syet}
180         ),
181         syet : COOPN!EventTerm(
182             expression <- 'this.fire' + h.source.fsm.FSMInv.name + '
                    RouterPost mname mparam mdirection'
183         ),
184         evt : COOPN!"COOPNMetamodel::ClassModule::Event"(
185             owenedEventTerm <- Set{evtet}
186         ),
187         evtet : COOPN!"COOPNMetamodel::ClassModule::EventTerm"(
188             expression <- 'fire' + h.source.fsm.FSMInv.name + 'Router' +
                    h.name + 'Pre mname mparam mdirection'
189         ),
190         cond : COOPN!"COOPNMetamodel::ClassModule::Condition"(
191             ownedEquations <- Set{eq,meq}
192         ),
193         eq : COOPN!"COOPNMetamodel::ClassModule::Equation"(
194             ownedOperatorEquation <- opeq,
195             ownedEquationTerms <- Set{lterm,rterm}
196         ),
197         lterm : COOPN!"COOPNMetamodel::ClassModule::Term"(
198             expression <- 'this'
199         ),
200         rterm : COOPN!"COOPNMetamodel::ClassModule::Term"(
201             expression <- 'Self'
202         ),
203         opeq : COOPN!"COOPNMetamodel::ClassModule::OperatorEquation"(
204             leftTerm <- lterm,
205             rightTerm <- rterm,
206             operator <- '='
207         ),
208         meq : COOPN!"COOPNMetamodel::ClassModule::Equation"(
209             ownedOperatorEquation <- mopeq,
210             ownedEquationTerms <- Set{mlterm,mrterm}
211         ),
212         mlterm : COOPN!"COOPNMetamodel::ClassModule::Term"(
213             expression <- 'mname'
214         ),
```

```
215         mrterm  : COOPN!"COOPNMetamodel::ClassModule::Term"(
216             expression <- h.name
217         ),
218         mopeq  : COOPN!"COOPNMetamodel::ClassModule::OperatorEquation"(
219             leftTerm <- mlterm,
220             rightTerm <- mrterm,
221             operator <- '='
222         )
223 }
```

D.20 Router05.atl

```
1  module Router05;
2  create coopnModel : COOPN refining hallModel : HALL, preCoopnModel :
      COOPN;
3
4  rule PackageRefactor{
5      from
6          s : COOPN!COOPNPackage
7      to
8          t : COOPN!COOPNPackage(
9              name <- s.name ,
10             ownedModules <- s.ownedModules
11         )
12 }
13
14 rule inherit{
15     from
16         s : COOPN!Inherit
17     to
18         t : COOPN!Inherit(
19         )
20 }
21
22 rule object{
23     from
24         s : COOPN!"COOPNMetamodel::ContextModule::Objects"
25     to
26         t : COOPN!"COOPNMetamodel::ContextModule::Objects"(
27             name <- s.name
28         )
29 }
30
31 rule sorts{
32     from
33         s : COOPN!"COOPNMetamodel::ADTModule::Sorts"
34     to
35         t : COOPN!"COOPNMetamodel::ADTModule::Sorts"(
36             name <- s.name
37         )
38 }
39
```

```
40  rule baseBodyRefactor{
41      from
42          s : COOPN!"COOPNMetamodel::ContextModule::Body"
43          (s.contextContainsBody.name = 'BaseContext')
44      to
45          t : COOPN!"COOPNMetamodel::ContextModule::Body"(
46                  ownedContextUses <- s.ownedContextUses,
47                  ownedAxiomTheorems <- s.ownedAxiomTheorems
48                      -> union(HALL!UserProfile.allInstancesFrom('hallModel
                            ')->select(h|not h.componentSet.first().
                            oclIsUndefined()))
49                      -> union(HALL!SystemComponent.allInstancesFrom('
                            hallModel')->select(h|not h.componentSet.first().
                            oclIsUndefined()))
50                      -> union(HALL!VisualObject.allInstancesFrom('hallModel
                            ')->select(h|not h.componentSet.first().
                            oclIsUndefined()))
51                      -> union(HALL!TaskObject.allInstancesFrom('hallModel')
                            ->select(h|not h.componentSet.first().
                            oclIsUndefined()))
52          )
53  }
54
55  rule userProfile2Axiom{
56      from
57          h : HALL!UserProfile
58          ( not h.componentSet.first().oclIsUndefined() )
59      to
60          c : COOPN!"COOPNMetamodel::ContextModule::Axiom"(
61              name <- h.name + 'up',
62              requiredEvent <- evt,
63              ownedCondition <- cond
64          ),
65          evt : COOPN!"COOPNMetamodel::ContextModule::RequiredEvent"(
66              ownedRequiredEventTerm <- Set{evtet}
67          ),
68          evtet : COOPN!"COOPNMetamodel::ContextModule::EventTerm"(
69              expression <- 'fire' + h.name + 'RouterPre mname mparam
                    mdirection'
70          ),
71          cond : COOPN!"COOPNMetamodel::ContextModule::Condition"(
72              ownedEquations <- Set{eq,deq}
73          ),
```

```
74        eq : COOPN!"COOPNMetamodel::ContextModule::Equation"(
75            ownedOperatorEquation <- opeq,
76            ownedEquationTerms <- Set{lterm,rterm}
77        ),
78        lterm : COOPN!"COOPNMetamodel::ContextModule::Term"(
79            expression <- 'this'
80        ),
81        rterm : COOPN!"COOPNMetamodel::ContextModule::Term"(
82            expression <- 'Self'
83        ),
84        opeq : COOPN!"COOPNMetamodel::ContextModule::OperatorEquation"(
85            leftTerm <- lterm,
86            rightTerm <- rterm,
87            operator <- '='
88        ),
89        deq : COOPN!"COOPNMetamodel::ContextModule::Equation"(
90            ownedOperatorEquation <- dopeq,
91            ownedEquationTerms <- Set{dlterm,drterm}
92        ),
93        dlterm : COOPN!"COOPNMetamodel::ContextModule::Term"(
94            expression <- 'mdirection'
95        ),
96        drterm : COOPN!"COOPNMetamodel::ContextModule::Term"(
97            expression <- 'true'
98        ),
99        dopeq : COOPN!"COOPNMetamodel::ContextModule::OperatorEquation
            "(
100           leftTerm <- dlterm,
101           rightTerm <- drterm,
102           operator <- '='
103       )
104 }
105
106 rule systemComponent2Axiom{
107     from
108         h : HALL!SystemComponent
109         ( not h.componentSet.first().oclIsUndefined() )
110
111     to
112         -- similar to rule userProfile2Axiom where UserProfile is
            replaced by SystemComponent
113 }
114
```

```
115 rule taskObject2Axiom{
116     from
117         h : HALL!TaskObject
118         ( not h.componentSet.first().oclIsUndefined() )
119     to
120         -- similar to rule userProfile2Axiom where UserProfile is
                replaced by TaskObject
121 }
122
123 rule visualObject2Axiom{
124     from
125         h : HALL!VisualObject
126         ( not h.componentSet.first().oclIsUndefined() )
127     to
128         -- similar to rule userProfile2Axiom where UserProfile is
                replaced by VisualObject
129 }
```

D.21 Router06.atl

```
1  module Router06;
2  create coopnModel : COOPN refining hallModel : HALL, preCoopnModel :
      COOPN;
3
4  rule PackageRefactor{
5      from
6          s : COOPN!COOPNPackage
7      to
8          t : COOPN!COOPNPackage(
9              name <- s.name ,
10             ownedModules <- s.ownedModules
11         )
12 }
13
14 rule inherit{
15     from
16         s : COOPN!Inherit
17     to
18         t : COOPN!Inherit(
19         )
20 }
21
22 rule object{
23     from
24         s : COOPN!"COOPNMetamodel::ContextModule::Objects"
25     to
26         t : COOPN!"COOPNMetamodel::ContextModule::Objects"(
27             name <- s.name
28         )
29 }
30
31 rule sorts{
32     from
33         s : COOPN!"COOPNMetamodel::ADTModule::Sorts"
34     to
35         t : COOPN!"COOPNMetamodel::ADTModule::Sorts"(
36             name <- s.name
37         )
38 }
39
```

```
40 rule baseBodyRefactor{
41     from
42         s : COOPN!"COOPNMetamodel::ContextModule::Body"
43         (s.contextContainsBody.name = 'BaseContext')
44     to
45         t : COOPN!"COOPNMetamodel::ContextModule::Body"(
46                 ownedContextUses <- s.ownedContextUses,
47                 ownedAxiomTheorems <- s.ownedAxiomTheorems
48                     -> union(HALL!UserProfile.allInstancesFrom('hallModel
                           ')->select(h|not h.componentSetInv.oclIsUndefined()
                           ))
49                     -> union(HALL!SystemComponent.allInstancesFrom('
                           hallModel')->select(h|not h.componentSetInv.
                           oclIsUndefined()))
50                     -> union(HALL!VisualObject.allInstancesFrom('hallModel
                           ')->select(h|not h.componentSetInv.oclIsUndefined()
                           ))
51                     -> union(HALL!TaskObject.allInstancesFrom('hallModel')
                           ->select(h|not h.componentSetInv.oclIsUndefined()))
52         )
53 }
54
55 rule axiomRefactor{
56     from
57         s : COOPN!"COOPNMetamodel::ContextModule::Axiom"
58     to
59         t : COOPN!"COOPNMetamodel::ContextModule::Axiom"(
60             name <- s.name,
61             providedEvent <- synch,
62             requiredEvent <- s.requiredEvent,
63             ownedCondition <- s.ownedCondition
64         ),
65         synch : COOPN!"COOPNMetamodel::ContextModule::ProvidedEvent"(
66             ownedProvidedEventTerm <- Set{HALL!UserProfile.
                 allInstancesFrom('hallModel')
67                         ->select(e| if e.componentSetInv.oclIsUndefined
                               ()
68                               then
69                                   false
70                               else
71                                   s.name = e.componentSetInv.name + 'up'
72                               endif )}
```

```
73                              ->union(Set{HALL!SystemComponent.
                                   allInstancesFrom('hallModel')
74                              ->select(e| if e.componentSetInv.oclIsUndefined
                                   ()
75                                 then
76                                    false
77                                 else
78                                    s.name = e.componentSetInv.name + 'up'
79                                 endif )})
80                              ->union(Set{HALL!TaskObject.allInstancesFrom('
                                   hallModel')
81                              ->select(e| if e.componentSetInv.oclIsUndefined
                                   ()
82                                 then
83                                    false
84                                 else
85                                    s.name = e.componentSetInv.name + 'up'
86                                 endif )})
87                              ->union(Set{HALL!VisualObject.allInstancesFrom('
                                   hallModel')
88                              ->select(e| if e.componentSetInv.oclIsUndefined
                                   ()
89                                 then
90                                    false
91                                 else
92                                    s.name = e.componentSetInv.name + 'up'
93                                 endif )})-- melhorar isto
94          )
95   }
96
97   rule userProfile2EventTerm{
98      from
99          h : HALL!UserProfile
100         ( not h.componentSetInv.oclIsUndefined() )
101     to
102         c : COOPN!"COOPNMetamodel::ContextModule::EventTerm"(
103             expression <- 'this.fire' + h.name + 'RouterPost mname
                    mparam mdirection'
104         )
105  }
106
107  rule systemComponent2EventTerm{
108     from
```

```
109           h : HALL!SystemComponent
110           ( not h.componentSetInv.oclIsUndefined() )
111       to
112           -- similar to rule userProfile2EventTerm
113   }
114
115   rule taskObject2EventTerm{
116       from
117           h : HALL!TaskObject
118           ( not h.componentSetInv.oclIsUndefined() )
119
120       to
121           -- similar to rule userProfile2EventTerm
122   }
123
124   rule visualObject2EventTerm{
125       from
126           h : HALL!VisualObject
127           ( not h.componentSetInv.oclIsUndefined() )
128
129       to
130           -- similar to rule userProfile2EventTerm
131   }
```

D.22 Router07.atl

```
1  module Router07;
2  create coopnModel : COOPN refining hallModel : HALL, preCoopnModel :
      COOPN;
3
4  rule PackageRefactor{
5      from
6          s : COOPN!COOPNPackage
7      to
8          t : COOPN!COOPNPackage(
9              name <- s.name ,
10             ownedModules <- s.ownedModules
11         )
12 }
13
14 rule inherit{
15     from
16         s : COOPN!Inherit
17     to
18         t : COOPN!Inherit(
19         )
20 }
21
22 rule object{
23     from
24         s : COOPN!"COOPNMetamodel::ContextModule::Objects"
25     to
26         t : COOPN!"COOPNMetamodel::ContextModule::Objects"(
27             name <- s.name
28         )
29 }
30
31 rule sorts{
32     from
33         s : COOPN!"COOPNMetamodel::ADTModule::Sorts"
34     to
35         t : COOPN!"COOPNMetamodel::ADTModule::Sorts"(
36             name <- s.name
37         )
38 }
39
```

```
40  rule baseBodyRefactor{
41      from
42          s : COOPN!"COOPNMetamodel::ContextModule::Body"
43          (s.contextContainsBody.name = 'BaseContext')
44      to
45          t : COOPN!"COOPNMetamodel::ContextModule::Body"(
46                  ownedContextUses <- s.ownedContextUses,
47                  ownedAxiomTheorems <- s.ownedAxiomTheorems
48                      -> union(HALL!UserProfile.allInstancesFrom('hallModel
                           ')->select(h|not h.componentSetInv.oclIsUndefined()
                           ))
49                      -> union(HALL!SystemComponent.allInstancesFrom('
                           hallModel')->select(h|not h.componentSetInv.
                           oclIsUndefined()))
50                      -> union(HALL!VisualObject.allInstancesFrom('hallModel
                           ')->select(h|not h.componentSetInv.oclIsUndefined()
                           ))
51                      -> union(HALL!TaskObject.allInstancesFrom('hallModel')
                           ->select(h|not h.componentSetInv.oclIsUndefined()))
52          )
53  }
54
55  rule userProfile2Axiom{
56      from
57          h : HALL!UserProfile
58          ( not h.componentSetInv.oclIsUndefined() )
59      to
60          c : COOPN!"COOPNMetamodel::ContextModule::Axiom"(
61              name <- h.name + 'down',
62              providedEvent <- synch,
63              requiredEvent <- evt,
64              ownedCondition <- cond
65          ),
66          synch : COOPN!"COOPNMetamodel::ContextModule::ProvidedEvent"(
67              ownedProvidedEventTerm <- Set{syet}
68          ),
69          syet : COOPN!EventTerm(
70              expression <- 'this.fire' + h.componentSetInv.name + '
                    RouterPost mname mparam mdirection'
71          ),
72          evt : COOPN!"COOPNMetamodel::ContextModule::RequiredEvent"(
73              ownedRequiredEventTerm <- Set{evtet}
74          ),
```

```
75          evtet : COOPN!"COOPNMetamodel::ContextModule::EventTerm"(
76              expression <- 'fire' + h.name + 'RouterPre mname mparam
                    mdirection'
77          ),
78          cond : COOPN!"COOPNMetamodel::ContextModule::Condition"(
79              ownedEquations <- Set{eq,deq}
80          ),
81          eq : COOPN!"COOPNMetamodel::ContextModule::Equation"(
82              ownedOperatorEquation <- opeq,
83              ownedEquationTerms <- Set{lterm,rterm}
84          ),
85          lterm : COOPN!"COOPNMetamodel::ContextModule::Term"(
86              expression <- 'this'
87          ),
88          rterm : COOPN!"COOPNMetamodel::ContextModule::Term"(
89              expression <- 'Self'
90          ),
91          opeq : COOPN!"COOPNMetamodel::ContextModule::OperatorEquation"(
92              leftTerm <- lterm,
93              rightTerm <- rterm,
94              operator <- '='
95          ),
96          deq : COOPN!"COOPNMetamodel::ContextModule::Equation"(
97              ownedOperatorEquation <- dopeq,
98              ownedEquationTerms <- Set{dlterm,drterm}
99          ),
100         dlterm : COOPN!"COOPNMetamodel::ContextModule::Term"(
101             expression <- 'mdirection'
102         ),
103         drterm : COOPN!"COOPNMetamodel::ContextModule::Term"(
104             expression <- 'false'
105         ),
106         dopeq : COOPN!"COOPNMetamodel::ContextModule::OperatorEquation
                "(
107             leftTerm <- dlterm,
108             rightTerm <- drterm,
109             operator <- '='
110         )
111 }
112
113 rule systemComponent2Axiom{
114     from
115         h : HALL!SystemComponent
```

```
116         ( not h.componentSetInv.oclIsUndefined() )
117
118     to
119         -- similar to rule userProfile2Axiom
120 }
121
122 rule taskObject2Axiom{
123     from
124         h : HALL!TaskObject
125         ( not h.componentSetInv.oclIsUndefined() )
126
127     to
128         -- similar to rule userProfile2Axiom
129 }
130
131 rule visualObject2Axiom{
132     from
133         h : HALL!VisualObject
134         ( not h.componentSetInv.oclIsUndefined() )
135
136     to
137         -- similar to rule userProfile2Axiom
138 }
```

E. XMI models used in testing

E.1 First Phase Test (H)ALL Source Model

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <HALL:Model xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
     xmlns:HALL="HALL">
3    <userProfile name="user01"/>
4    <systemComponent name="sistem01" componentSet="//@systemComponent.1
       ">
5      <data name="Data01"/>
6      <FSM>
7        <initialState>
8          <transitions name="toState01" stateRef="//@systemComponent.0/
             @FSM/@state.0">
9            <PreCondition/>
10           <PosCondition/>
11           <Action/>
12           <Trigger/>
13         </transitions>
14       </initialState>
15       <state name="State01"/>
16     </FSM>
17     <messageHandlerSet name="Message␣Handler01"/>
18   </systemComponent>
19   <systemComponent name="sistem02" componentSetInv="//
       @systemComponent.0"/>
20   <messageDefinition name="mesage01"/>
21 </HALL:Model>
```

E.2 Example of CO-OPN output

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <COOPNMetamodel:COOPNPackage xmi:version="2.0" xmlns:xmi="http://www.
       omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:COOPNMetaModel.ContextModule="ContextModule.ecore"
       xmlns:COOPNMetamodel="http:///COOPNMetamodel.ecore" name="
       HALLModelPackage">
3    <ownedModules xsi:type="COOPNMetaModel.ContextModule:COOPNContext"
         name="UserProfileuser01" contextUse="//@ownedModules.2/
         @ownedBody/@ownedContextUses.0">
4      <ownedBody/>
5      <ownedInherits inheritedContext="//@ownedModules.7"/>
6    </ownedModules>
7    <ownedModules xsi:type="COOPNMetaModel.ContextModule:COOPNContext"
         name="SystemComponentsistem02" contextUse="//@ownedModules.2/
         @ownedBody/@ownedContextUses.2">
8      <ownedBody/>
9      <ownedInherits inheritedContext="//@ownedModules.5"/>
10   </ownedModules>
11   <ownedModules xsi:type="COOPNMetaModel.ContextModule:COOPNContext"
         name="BaseContext">
12     <ownedBody>
13       <ownedContextUses usedContext="//@ownedModules.0"/>
14       <ownedContextUses usedContext="//@ownedModules.8"/>
15       <ownedContextUses usedContext="//@ownedModules.1"/>
16     </ownedBody>
17   </ownedModules>
18   <ownedModules xsi:type="COOPNMetaModel.ContextModule:COOPNContext"
         name="GenericVisualObject">
19     <ownedInherits inheritedContext="//@ownedModules.4"/>
20   </ownedModules>
21   <ownedModules xsi:type="COOPNMetaModel.ContextModule:COOPNContext"
         name="GenericComponent" inherit="//@ownedModules.7/
         @ownedInherits.0␣//@ownedModules.5/@ownedInherits.0␣//
         @ownedModules.3/@ownedInherits.0␣//@ownedModules.6/
         @ownedInherits.0"/>
22   <ownedModules xsi:type="COOPNMetaModel.ContextModule:COOPNContext"
         name="GenericSystemComponent" inherit="//@ownedModules.8/
         @ownedInherits.0␣//@ownedModules.1/@ownedInherits.0">
23     <ownedInherits inheritedContext="//@ownedModules.4"/>
24   </ownedModules>
```

```
25   <ownedModules xsi:type="COOPNMetaModel.ContextModule:COOPNContext"
         name="GenericTaskObject">
26     <ownedInherits inheritedContext="//@ownedModules.4"/>
27   </ownedModules>
28   <ownedModules xsi:type="COOPNMetaModel.ContextModule:COOPNContext"
         name="GenericUserProfile" inherit="//@ownedModules.0/
         @ownedInherits.0">
29     <ownedInherits inheritedContext="//@ownedModules.4"/>
30   </ownedModules>
31   <ownedModules xsi:type="COOPNMetaModel.ContextModule:COOPNContext"
         name="SystemComponentsistem01" contextUse="//@ownedModules.2/
         @ownedBody/@ownedContextUses.1">
32     <ownedBody/>
33     <ownedInherits inheritedContext="//@ownedModules.5"/>
34   </ownedModules>
35 </COOPNMetamodel:COOPNPackage>
```

E.3 Case Study (H)ALL Source Model

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <HALL:Model xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:HALL="
       HALL">
3    <userProfile name="Expert">
4      <visualObject name="MainPanel" componentSet="//@userProfile.0/
           @visualObject.1"/>
5      <visualObject name="SplitPanel" componentSet="//@userProfile.0/
           @visualObject.2␣//@userProfile.0/@visualObject.3"
           componentSetInv="//@userProfile.0/@visualObject.0"/>
6      <visualObject name="TreePanel" componentSetInv="//@userProfile.0/
           @visualObject.1"/>
7      <visualObject name="TabbedPanel" componentSet="//@userProfile.0/
           @visualObject.4␣//@userProfile.0/@visualObject.5"
           componentSetInv="//@userProfile.0/@visualObject.1"/>
8      <visualObject name="StatusPanel" componentSetInv="//@userProfile
           .0/@visualObject.3"/>
9      <visualObject name="CommandPanel" componentSet="//@userProfile.0/
           @visualObject.6␣//@userProfile.0/@visualObject.8␣//
           @userProfile.0/@visualObject.10" componentSetInv="//
           @userProfile.0/@visualObject.3"/>
10     <visualObject name="LabelPanel" componentSet="//@userProfile.0/
           @visualObject.7" componentSetInv="//@userProfile.0/
           @visualObject.5"/>
11     <visualObject name="Label01" componentSetInv="//@userProfile.0/
           @visualObject.6"/>
12     <visualObject name="UpPanel" componentSet="//@userProfile.0/
           @visualObject.9" componentSetInv="//@userProfile.0/
           @visualObject.5"/>
13     <visualObject name="UpButton01" componentSetInv="//@userProfile
           .0/@visualObject.8"/>
14     <visualObject name="DownPanel" componentSet="//@userProfile.0/
           @visualObject.11" componentSetInv="//@userProfile.0/
           @visualObject.5"/>
15     <visualObject name="DownButton01" componentSetInv="//@userProfile
           .0/@visualObject.10"/>
16     <taskObject name="SelectTreeNode" componentSet="//@userProfile.0/
           @taskObject.1␣//@userProfile.0/@taskObject.2"/>
17     <taskObject name="ButtonClick" componentSetInv="//@userProfile.0/
           @taskObject.0"/>
```

```
18      <taskObject name="RefreshGui" componentSet="//@userProfile.0/
            @taskObject.3␣//@userProfile.0/@taskObject.4" componentSetInv=
            "//@userProfile.0/@taskObject.0"/>
19      <taskObject name="RefreshControllerTree" componentSetInv="//
            @userProfile.0/@taskObject.2"/>
20      <taskObject name="RefreshCommandPanel" componentSetInv="//
            @userProfile.0/@taskObject.2"/>
21    </userProfile>
22    <systemComponent name="Partition" componentSet="//@systemComponent
        .1␣//@systemComponent.3␣//@systemComponent.6␣//@systemComponent
        .8␣//@systemComponent.9">
23      <FSM>
24        <initialState>
25          <transitions name="GotoNone" stateRef="//@systemComponent.0/
              @FSM/@state.0"/>
26        </initialState>
27        <state name="None">
28          <transitions name="GotoBooted" stateRef="//@systemComponent
              .0/@FSM/@state.1"/>
29        </state>
30        <state name="Booted">
31          <transitions name="GotoNone" stateRef="//@systemComponent.0/
              @FSM/@state.0"/>
32          <transitions name="GotoInitial" stateRef="//@systemComponent
              .0/@FSM/@state.2">
33            <PreCondition>
34              <PreConditionSet xsi:type="HALL:PreCEVarRef" name="
                  ButtonID" type="String"/>
35              <PreConditionSet xsi:type="HALL:PreCEBinaryOperator"
                  rightexpression="//@systemComponent.0/@FSM/@state.1/
                  @transitions.1/@PreCondition/@PreConditionSet.2"
                  leftexpression="//@systemComponent.0/@FSM/@state.1/
                  @transitions.1/@PreCondition/@PreConditionSet.0"
                  operatorname="Equals"/>
36              <PreConditionSet xsi:type="HALL:PreCEVarRef" name="Down03
                  " type="String"/>
37            </PreCondition>
38            <Action>
39              <ActionExpressionSet xsi:type="HALL:AEMessageInvocation"
                  name="ChangeStateToInitial" isTopDown="true"/>
40            </Action>
41            <Trigger>
```

```
42              <TriggerExpressionSet xsi:type="HALL:DomainEventFired"
                    String="ButtonClicked"/>
43            </Trigger>
44          </transitions>
45        </state>
46        <state name="Initial">
47          <transitions name="GotoNone" stateRef="//@systemComponent.0/
                @FSM/@state.0"/>
48          <transitions name="GotoBooted" stateRef="//@systemComponent
                .0/@FSM/@state.1"/>
49          <transitions name="GotoConnected" stateRef="//
                @systemComponent.0/@FSM/@state.4"/>
50          <transitions name="GotoConfigured" stateRef="//
                @systemComponent.0/@FSM/@state.3"/>
51        </state>
52        <state name="Configured">
53          <transitions name="GotoConnected" stateRef="//
                @systemComponent.0/@FSM/@state.4"/>
54          <transitions name="GotoNone" stateRef="//@systemComponent.0/
                @FSM/@state.0"/>
55        </state>
56        <state name="Connected">
57          <transitions name="GotoReady" stateRef="//@systemComponent.0/
                @FSM/@state.12"/>
58          <transitions name="GotoRunning" stateRef="//@systemComponent
                .0/@FSM/@state.14"/>
59          <transitions name="GotoNone" stateRef="//@systemComponent.0/
                @FSM/@state.0"/>
60        </state>
61        <state name="GTHstopped">
62          <transitions name="GotoConnected" stateRef="//
                @systemComponent.0/@FSM/@state.4"/>
63          <transitions name="GotoNone" stateRef="//@systemComponent.0/
                @FSM/@state.0"/>
64        </state>
65        <state name="SFOStopped">
66          <transitions name="GotoGTHStoped" stateRef="//
                @systemComponent.0/@FSM/@state.5"/>
67          <transitions name="GotoNone" stateRef="//@systemComponent.0/
                @FSM/@state.0"/>
68        </state>
69        <state name="EFStopped">
```

```
70          <transitions name="GotoSFOStopped" stateRef="//
                @systemComponent.0/@FSM/@state.6"/>
71          <transitions name="GotoNone" stateRef="//@systemComponent.0/
                @FSM/@state.0"/>
72        </state>
73        <state name="EBStoped">
74          <transitions name="GotoEFStopped" stateRef="//
                @systemComponent.0/@FSM/@state.7"/>
75          <transitions name="GotoNone" stateRef="//@systemComponent.0/
                @FSM/@state.0"/>
76        </state>
77        <state name="L2Stopped">
78          <transitions name="GotoEBStopped" stateRef="//
                @systemComponent.0/@FSM/@state.8"/>
79          <transitions name="GotoNone" stateRef="//@systemComponent.0/
                @FSM/@state.0"/>
80        </state>
81        <state name="L2SVStopped">
82          <transitions name="GotoL2Stopped" stateRef="//
                @systemComponent.0/@FSM/@state.9"/>
83          <transitions name="GotoNone" stateRef="//@systemComponent.0/
                @FSM/@state.0"/>
84        </state>
85        <state name="ROIBStoped">
86          <transitions name="GotoL2SVStopped" stateRef="//
                @systemComponent.0/@FSM/@state.10"/>
87          <transitions name="GotoNone" stateRef="//@systemComponent.0/
                @FSM/@state.0"/>
88        </state>
89        <state name="Ready">
90          <transitions name="GotoRunning"/>
91          <transitions name="GotoROIBStopped" stateRef="//
                @systemComponent.0/@FSM/@state.11"/>
92          <transitions name="GotoNone" stateRef="//@systemComponent.0/
                @FSM/@state.0"/>
93        </state>
94        <state name="Paused">
95          <transitions name="GotoRunning" stateRef="//@systemComponent
                .0/@FSM/@state.14"/>
96          <transitions name="GotoNone" stateRef="//@systemComponent.0/
                @FSM/@state.0"/>
97        </state>
98        <state name="Running">
```

```
99          <transitions name="GotoPaused" stateRef="//@systemComponent
                .0/@FSM/@state.13"/>
100         <transitions name="GotoNone" stateRef="//@systemComponent.0/
                @FSM/@state.0"/>
101       </state>
102     </FSM>
103     <messageHandlerSet name="FaultState">
104       <messageState name="FaultStateMessageHandled"/>
105       <initialMessageState>
106         <transitions name="ProcessFaultState" stateRef="//
                @systemComponent.0/@messageHandlerSet.0/@messageState.0"/>
107       </initialMessageState>
108     </messageHandlerSet>
109     <messageHandlerSet name="ChangeState">
110       <messageState name="SelfUpdateHandled"/>
111       <messageState name="ProcessUpdateHandled"/>
112       <initialMessageState>
113         <transitions name="ProcessUpdate" stateRef="//
                @systemComponent.0/@messageHandlerSet.1/@messageState.1"/>
114         <transitions name="UpdateSelfState" stateRef="//
                @systemComponent.0/@messageHandlerSet.1/@messageState.0"/>
115       </initialMessageState>
116     </messageHandlerSet>
117     <messageHandlerSet name="InformState">
118       <messageState name="InformStateMessageHandled"/>
119       <initialMessageState>
120         <transitions name="ProcessInformState" stateRef="//
                @systemComponent.0/@messageHandlerSet.2/@messageState.0"/>
121       </initialMessageState>
122     </messageHandlerSet>
123   </systemComponent>
124   <systemComponent name="SegmentNode1" componentSet="//
        @systemComponent.2" componentSetInv="//@systemComponent.0"/>
125   <systemComponent name="ApplicationNode1" componentSetInv="//
        @systemComponent.1"/>
126   <systemComponent name="SegmentNode2" componentSet="//
        @systemComponent.4␣//@systemComponent.5" componentSetInv="//
        @systemComponent.0"/>
127   <systemComponent name="ApplicationNode2" componentSetInv="//
        @systemComponent.3"/>
128   <systemComponent name="ResourceNode1" componentSetInv="//
        @systemComponent.3"/>
```

```
129    <systemComponent name="SegmentNode3" componentSet="//
           @systemComponent.7" componentSetInv="//@systemComponent.0"/>
130    <systemComponent name="ResourceNode2" componentSetInv="//
           @systemComponent.6"/>
131    <systemComponent name="SegmentNode4" componentSet="//
           @systemComponent.13" componentSetInv="//@systemComponent.0"/>
132    <systemComponent name="SegmentNode5" componentSet="//
           @systemComponent.14" componentSetInv="//@systemComponent.0"/>
133    <systemComponent name="ResourceNode3" componentSetInv="//
           @systemComponent.13"/>
134    <systemComponent name="ResourceNode4" componentSetInv="//
           @systemComponent.13"/>
135    <systemComponent name="ApplicationNode3" componentSetInv="//
           @systemComponent.13"/>
136    <systemComponent name="SegmentNode6" componentSet="//
           @systemComponent.10␣//@systemComponent.11␣//@systemComponent.12"
            componentSetInv="//@systemComponent.8"/>
137    <systemComponent name="ResourceNode5" componentSetInv="//
           @systemComponent.9"/>
138    <messageDefinition name="FaultState">
139      <parameter name="ComponentName" type="String"/>
140      <parameter name="StateName" type="String"/>
141    </messageDefinition>
142    <messageDefinition name="InformState">
143      <parameter name="ComponentName" type="String"/>
144      <parameter name="StateName" type="String"/>
145    </messageDefinition>
146    <messageDefinition name="ChangeState">
147      <parameter name="ComponentName" type="String"/>
148      <parameter name="StateName" type="String"/>
149    </messageDefinition>
150  </HALL:Model>
```

Bibliography

[1] V. Amaral B. Barroca. (H)ALL: a DSVL for designing user interfaces for control systems. In *5th Nordic Workshop on Model Driven Engineering NW-MoDE*, pages 27–29. Blekinge Institute of Technology, aug 2007. http://www.ituniv.se/~miroslaw/node.htm.

[2] Bruno Barroca. (H)ALL: a DSL for rapid prototyping of user interfaces for control systems. Master's thesis, Faculdade de Ciências da Universidade Nova de Lisboa, 2007.

[3] Achim D. Brucker and Burkhart Wol. A proposal for a formal ocl semantics in isabelle/hol. In *Theorem Proving in Higher Order Logics, LNCS 2410*, pages 99–114. Springer, 2002.

[4] Claudia Pons Carlos G. Neil. Formalizing the model transformation using metamodeling techniques. In *Proc. Argentine Symposium on Software Engineering. Jornadas Argentinas de Informática e Investigación Operativa*, 2004. http://www.lifia.info.unlp.edu.ar/papers/2004/Claudia2004.pdf.

[5] N. Guelfi D. Buchs. A formal specification framework for object-oriented distributed systems. *IEEE Trans. Software Eng.*, 26(7):635–652, 2000.

[6] András Balogh Dániel Várró. The model transformation language of the viatra2 framework. In *Science of Computer Programing 68(2007)*, pages 214–234, 2007.

[7] Heiko Dörr. *Efficient Graph Rewriting and its Implementations*, volume 922 of *Lecture Notes in Computer Science*. SPRINGER, 1995.

[8] Institute for Software Integrated Systems. Generic modeling environment. http://www.isis.vanderbilt.edu/projects/gme/, 2007.

[9] The Eclipse Foundation. ATL/User Guide. http://wiki.eclipse.org/ATL/User_Guide.

[10] The Eclipse Foundation. Gmf project page. http://www.eclipse.org/modeling/gmf/.

[11] David S. Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing*. John Wiley & Sons, 2003.

[12] Ivan Kurtev Frédéric Jouault. Transforming models with ATL. In *Model Transformations in Practice Workshop at MoDELS 2005*, Montego Bay, Jamaica, 2005. http://sosym.dcs.kcl.ac.uk/events/mtip/submissions/jouault_kurtev__transforming_models_with_atl.pdf.

[13] Ivan Kurtev Frédéric Jouault. On the architectural alignment of ATL and QVT. In *2006 ACM Symposium on Applied Computing (SAC 06)*, pages pages 1188–1195, Dijon, France, 2006. ACM Press. http://www.sciences.univ-nantes.fr/lina/atl/www/papers/ATLandQVT-PRELIMINARY%20VERSION.pdf.

[14] Object Management Group. Metaobject facility. http://www.omg.org/mof/.

[15] Object Management Group. Object constraint language specification. http://www.omg.org/technology/documents/formal/ocl.htm.

[16] Object Management Group. Object management group. http://www.omg.org/.

[17] Object Management Group. Unified modeling language. http://www.uml.org/.

[18] SMV group. COOPNBuilder IDE tools. http://smv.unige.ch/tiki-index.php?page=IntroCoopn.

[19] ATLAS group LINA & INRIA. ATL:Atlas Transformation Language,Specification of the ATL Virtual Machine. http://www.eclipse.org/m2m/atl/doc/ATL_VMSpecification%5Bv00.01%5D.pdf.

[20] Anja Eline Bekkelien Joachim Flammer, Mihai Caprini. IGUI online help. `http://atlas-onlsw.web.cern.ch/Atlas-onlsw/components/igui/tdaq-02-00-00/online-help/Introduction.htm`.

[21] Juan de Lara Laszló Lengyel Tihamér Levendovszky Ulrike Prange Gabriele Taentzer Dániel Varró Szilvia V. Gyapay Karsten Ehrig, Esther Guerra. Model transformation by graph transformation: A comparative study. In *MTiP '05: Proceedings of the International Workshop on Model Transformations in Practice*, 2005.

[22] Simon Helsen Krzysztof Czarnecki. Classification of model transformation approaches. In *OOPSLA 2003 Workshop on Generative Techniques in the context of Model Driven Architecture*, oct 2003. `http://www.softmetaware.com/oopsla2003/czarnecki.pdf`.

[23] Ivan Kurtev, Klaas van den Berg, and Frédéric Jouault. Evaluation of rule-based modularization in model transformation languages illustrated with atl. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 1202–1209, New York, NY, USA, 2006. ACM Press.

[24] Jochen M. Kuster. Systematic validation of model transformations. In *Essentials of the 3rd UML Workshop in Software Model Engineering (WiSME2́004)*, 2004.

[25] Maher Lamari. Towards an automated test generation for the verification of model transformations. In *Proceedings of the 2007 ACM symposium on Applied computing*, pages 998–1005, Seoul, Korea, 2007.

[26] Levi Lucio, Luis Pedro, and Didier Buchs. A test language for co-opn specifications. In *RSP '05: Proceedings of the 16th IEEE International Workshop on Rapid System Prototyping (RSP'05)*, pages 195–201, Washington, DC, USA, 2005. IEEE Computer Society.

[27] Didier Buchs Luis Pedro, Levi Lucio. System prototype and verification using metamodel-based transformations. *IEEE Distributed Systems Online*, vol. 8(no. 4), 2007. art. no. 0704-o4001.

[28] Microsoft. Msdn:domain-specific language tools. `http://msdn2.microsoft.com/en-us/vstudio/aa718368.aspx`, 2007.

[29] N. Guelfi O. Biberstein, D. Buchs. Object-oriented nets with algebraic specifications: The co-opn/2 formalism. In *Concurrent Object-Oriented Programming and Petri Nets*, pages 70–130, 2001.

[30] BATIC^{3}S project. BATIC^{3}S - bibliography. `http://smv.unige.ch/tiki-index.php?page=BATICSBiblio`.

[31] QVT-Partners. Qvt-partners. `http://qvtp.org`.

[32] Vasco Sousa. A visual domain specific language for modeling high energy physics queries. a comparative study of technologies and pheasant implementation. Technical report, Faculdade de Ciências da Universidade Nova de Lisboa, 2007.

[33] Markus Völter Tom Stahl. *Model-Driven Software Development - Technology, Engineering, Management*. Wiley, 2006.

[34] J. Wang, S.-K. Kim, and D. Carrington. Verifying metamodel coverage of model transformations. In *Proceedings of the 2006 Australian Software Engineering Conference (ASWEC0́6)*, 2006.