Technical University of Crete, Greece
School of Electrical and Computer Engineering

An Integrated Development Environment for the ASEME Methodology

Floros Efthymios

Thesis Committee
Associate Professor Georgios Chalkiadakis
Associate Professor Michail G. Lagoudakis
Dr. Nikolaos Spanoudakis

Chania, October 2017

Πολυτεχνείο Κρήτης
Τμήμα Ηλεκτρονικών Μηχανικών και Μηχανικών Υπολογιστών

Ολοκληρωμένο Περιβάλλον Ανάπτυξης Λογισμικού για την Μεθοδολογία ASEME

Φλώρος Ευθύμιος

Εξεταστική Επιτροπή
Αναπληρωτής Καθηγητής Γεώργιος Χαλκιαδάκης
Αναπληρωτής Καθηγητής Μιχαήλ Γ. Λαγουδάκης
Δρ. Νικόλαος Σπανουδάκης

Χανιά , Οκτώβριος 2017

Acknowledgements

First of all I would like to thank my family for their endless support and love. Next i would like to thank my supervisors Professor Georgios Chalkiadakis and Dr. Nikos Spanoudakis firstly for introducing me into the research field of intelligent agents and secondly for their guidance; and Professor Michail Lagoudakis for accepting to be at the jury committee. I also want to thank Shabana Shaikh for her contribution to ASEME IDE. Last but not least I want to thank the friends I made during my studies, who made me consider Chania as my second hometown.

Abstract

The design of multi-agent systems is a time-consuming task even for experts. ASEME is an Agent-Oriented Software Engineering Methodology that can be used for model-driven Agent System development. This thesis presents a complete Integrated Development Environment (IDE), by integrating together the various existing tools, changing and enriching models end enhancing the model driven process, which allows a user to design a multi-agent system from scratch, export it to business models(XPDL/BMPN) and generate Java and C++ code. This thesis also demonstrates the ASEME model-driven process with a simple negotiation Agent.

Περίληψη

Ο σχεδιασμός πολυπρακτορικών συστημάτων είναι χρονοβόρα διαδικασία ακόμη και για τους ειδικούς. Η ASEME είναι μια μεθοδολογία πρακτοροστρεφούς μηχανικής λογισμικού που μπορεί να χρησιμοποιηθεί για την ανάπτυξη πολυπρακτορικών συστημάτων με γνώμονα το μοντέλο. Με την ενσωμάτωση των διαφόρων υφιστάμενων εργαλείων, την αλλαγή και τον εμπλουτισμό των μοντέλων η εργασία αυτή παρουσιάζει ένα ολοκληρωμένο περιβάλλον αναπτυξης λογισμικού το οποίο επιτρέπει σε ένα χρήστη να σχεδιάσει ένα πολυ-πρακτορικό σύστημα από το μηδέν, να το εξάγει σε μοντέλα διαδικασιών (XPDL / BMPN) και να δημιουργήσει κώδικα Java και C ++. Αυτή η εργασία παρουσιάζει επίσης τη διαδικασία μετασχηματισμού μοντέλων της ASEME με έναν απλό πράκτορα διαπραγμάτευσης.

Contents

List of Figures

Chapter 1

Introduction

1.1 Problem Statement

Artificial intelligence, defined as intelligence exhibited by machines, has many applications in today's society, like Internet industry, health industry and many more. The artificial intelligence programs that take decisions in order to reach a user defined goal are called agents[23]. Agents should be able to interact with other agents -constituting a multi-agent system- in terms of cooperation or even the control of other agents, so while these interactions become more common every day the need to use multi-agent systems increases [22].

However, multi-agent systems can become really complicated as the number of agents increases, so a modular design approach is the solution. With model driven engineering, a simpler, easier and sometimes even more comprehensive approach to software development is provided [4]. In model driven software engineering the need of model transformations during the different development phases is essential. One model driven engineering methodology is ASEME ([15]). ASEME is an Agent-Oriented Software Engineering (AOSE) methodology for developing multi-agent systems. It uses the Agent Modeling Language (AMOLA, [16]), which provides the syntax and semantics for creating models of multi–agent systems covering the analysis and design phases of a software development process.

For some years now some stand alone tools have been developed based on parts of ASEME methodology and extending it such as the AMOLA metamodels and transformation tools[14], a CASE (Computer-Aided Software En-

gineering) tool for robot-team behavior-control development (Kouretes Statechart Editor - KSE) [20], [21], an extension of KSE for Executing Statechart-Based Robotic Behavior Models [9], [10] an application to transform SRM to BPMN/XPDL models [6], [7]. In this thesis we gathered together all the existing standalone tools, improved them and extended them in terms of functionality in order to develop a tool, more like an Integrated Development Environment, that allows a user to use the ASEME methodology at its full extend in order to design a multi-agent system.

1.2 Thesis Outline

As mentioned already this thesis contribution is a tool, or rather an Integrated Development Environment (IDE) that provides the user the possibility of designing a multi-agent system (MAS) using the ASEME methodology from scratch, and going through the updated ASEME model driven process to either export the designed system to business models (BPMN/XPDL) or code generation (Java/C++) or both.

In Chapter 2 we briefly present the concepts and tools used throughout this thesis, starting with a brief presentation of Model-Driven Engineering and its basis, Meta-modeling and Model Transformation. We continue with the presentation of the Eclipse Modeling Framework and the Eclipse Rich Client Platform that are the two axes of ASEME IDE development. After that we present the Xpand Language, a template language we used for code generation and the Java Agent Development Framework (JADE) which we use for the generated Java code. Subsequently we present the ASEME methodology, through an overview of ASEME process, a brief presentation of each of the AMOLA meta-models and of the Model-Driven-Process. Finally we present an overview on BPMN and XPDL which are the business models used.

In Chapter 3 we describe the implementation process of ASEME IDE. We begin with the Dashboard presenting the layout, architecture and the benefits of using a dashboard in an application in general. Afterwards we present and discuss each AMOLA meta-model separately presenting in more detail any changes and additions made to it and its editor and finally the functionality additions we made to the whole process including the replacement of IAC meta-model with Statechart[21], the new SRM to BPMN and IAC to GGenerator transformations available that extended the model- driven pro-

cess, the support of Abstract Roles and Protocols and many transformation refinements made. Some of these transformation refinements were obligatory due to the meta-models that changed, and others were made for the improvement of ASEME IDE.

In Chapter 4 we demonstrate the development process of adding new functionality to the existing tool using as example the addition of SRM to BPMN/XPDL tool[7]. In Chapter 5 we present an example on how to use the ASEME IDE going through each step of the Model-Driven process with detailed comments on the usage of the current version of the IDE and discussing what the user should note about the behavior of the tool.

Finally in Chapter 6, we outline the conclusions of this thesis, along with our ideas and plans for future work, discussing details of the ASEME IDE development process, features that did not have the time to complete, and providing advice based on the know-how we acquired completing this thesis.

Chapter 2

Background

2.1 Agent Oriented Software Engineering

Designing and building high quality industrial-strength software is difficult. Indeed, it has been claimed that such development projects are among the most complex construction tasks undertaken by humans. Against this background, a wide range of software engineering paradigms have been devised (e.g., object-oriented programming, design patterns, application frameworks etc.). Each successive development either claims to make the engineering process easier or to extend the complexity of applications that can feasibly be built.

Designing and implementing software as a collection of interacting, autonomous agents (Agent is an abstraction of autonomous software that is pro-active, social and can undertake tasks that if humans do them they require intelligence[22]) represents a promising point of departure for software engineering for certain classes of problem adopting a multi-agent approach to system development affords software engineers a number of significant advantages over contemporary methods.

The most important outstanding issues for agent-based software engineering are: (i) an understanding of the situations in which agent solutions are appropriate; and (ii) principled but informal development techniques for agent systems [5].

Towards the second issue AOSE proposes methodologies, abstractions and tools for developing systems based on the agent paradigm. Agent-based systems have evolved during the last two decades. To support the devel-

opment of such systems, agent-oriented methodologies have emerged. In general, most of the methodologies have originated from two major research domains, namely software engineering and artificial intelligence, and were adjusted to address the agent abstraction. It seems that many of the methodologies share a common basis, an observation that calls for unification and for standardization. [18]

2.2 Model Driven Engineering

Over the last few decades, software researchers and developers have been creating abstractions that help them program in terms of their design intent rather than the underlying computing environment — for example, CPU, memory and network devices — and at the same time shield them from the complexities of these environments. These model-driven engineering technologies offer a promising approach to address the inability of third-generation languages to alleviate the complexity of platforms and express domain concepts effectively[11]. Model-driven engineering (MDE) is a software development methodology that focuses on creating and exploiting domain models, which are conceptual models of all the topics related to a specific problem. Hence, it highlights and aims at abstract representations of the knowledge and activities that govern a particular application domain, rather than the computing (f.e. algorithmic) concepts.

The MDE approach is meant to increase productivity by maximizing compatibility between systems (via reuse of standardized models), simplifying the process of design (via models of recurring design patterns in the application domain), and promoting communication between individuals and teams working on the system (via a standardization of the terminology and the best practices used in the application domain).

A modeling paradigm for MDE is considered effective if its models make sense from the point of view of a user that is familiar with the domain, and if they can serve as a basis for implementing systems. The models are developed through extensive communication among product managers, designers, developers and users of the application domain. As the models approach completion, they enable the development of software and systems.

Some of the better known MDE initiatives are:

- the Object Management Group (OMG) initiative model-driven architecture (MDA), which is a registered trademark of OMG.

- the Eclipse ecosystem of programming and modeling tools (Eclipse Modeling Framework).

2.3 Metamodeling and Models Transformation

Model Driven Engineering (MDE) relies heavily on model transformation. Model transformation is the process of transforming a model to another model. The requirements for achieving the transformation are the existence of metamodels of the models in question and a transformation language in which to write the rules for transforming the elements of one metamodel to those of another metamodel. The MDE approach has been argued to contribute to non-functional requirements capture, such as portability, interoperability and reusability[1]. In the software engineering domain a model is an abstraction of a software system (or part of it) and a metamodel is another abstraction, defining the properties of the model itself. However, even a metamodel is itself a model. In the context of model engineering there is yet another level of abstraction, the metametamodel, which is defined as a model that conforms to itself. There are four types of model transformation techniques [17]:

- **Model to Model (M2M)** transformation. This kind of transformation is used for transforming a type of graphical model to another type of graphical model. A M2M transformation is based on the source and target metamodels and defines the transformations of elements of the source model to elements of the target model.

- **Text to Model (T2M)** transformation. This kind of transformation is used for transforming a textual representation to a graphical model. The textual representation must adhere to a language syntax definition usually using BNF. The graphical model must have a metamodel. Then, a transformation of the text to a graphical model can be defined.

- **Model to Text (M2T)** transformations. Such transformations are used for transforming a visual representation to code (code is text). Again, the syntax of the target language must be defined along with the metamodel of the graphical model.

- **Text to Text (T2T)** transformations. Such transformations are used for transforming a textual representation to another textual representation. This is usually the case when a program written for a specific programming language is transformed to a program in another programming language (e.g. a compiler).

2.4 Eclipse Modeling Project

In the heart of the model transformation procedure is the Eclipse Modeling Framework. Ecore is EMF's model of a model (metamodel). It functions as a metametamodel and it is used for constructing metamodels. It defines that a model is composed of instances of the EClass type, which can have attributes (instances of the EAttribute type) or reference other EClass instances (through the EReference type). Finally, EAttributes can be of various EDataType instances (such are integers, strings, etc). EMF allows to extend existing models via inheritance, using the ESuperType relationship for extending an existing EClass.

The Eclipse Modeling Project (EMP) [3] is a top-level project at Eclipse. In contrast, the core of the project, EMF, has been in existence for as long as the Eclipse platform itself. Today the modeling project is a collection of projects related to modeling technologies. This collection was formed to coordinate and focus on model-driven software development capabilities within Eclipse. The introduction of the Amalgamation project (`https://projects.eclipse.org/projects/modeling.amalgam`) ushered in the beginnings of a Domain Specific Language (DSL) focused development environment, although it has a long way to go before mainstream developers can use it. The Modeling project is organized logically into projects that provide the following capabilities: abstract syntax development, concrete syntax development, model-to-model transformation, and model-to-text transformation. A single project, the Model Development Tools (MDT) project, is dedicated to the support of industry-standard models. Another project within the Modeling project focuses on research in generative modeling technologies. Specifically EMP consists of EMF (Eclipse Modeling Framework), QVT (Query: Validation: Transaction), M2M (Model-to-Model transformation), M2T (Model- to-Text transformation), TMF (Textual Modeling Framework) and GMF (Graphical Modeling Framework). EMF allows the developer to define a DSL language in an abstract syntax. EMF has as an output a model

that describes a new language. QVT provides query, validation and transaction features for the EMF models. M2M provides Operational Mapping Language that allows model- to-model transformation for EMF models. M2T allows model-to-text by using JET (Java Emitter Template) or Xpand as a template engine. TMF is still under development and does not offer a lot of capabilities, but its purpose is to provide a textual editors with syntax highlighting, code completion and build for EMF models. In the other hand, GMF provides graphical editors for EMF models. The Graphical Modeling Framework (GMF) is a framework within the Eclipse platform. It provides a generative component and runtime infrastructure for developing graphical editors based on the Eclipse Modeling Framework (EMF) and Graphical Editing Framework (GEF). The project aims to provide these components, in addition to exemplary tools for select domain models which illustrate its capabilities.

2.5 Eclipse Rich Client Platform

While the Eclipse platform is designed to serve as an open tools platform, it is architected so that its components could be used to build just about any client application. The minimal set of plug-ins needed to build a rich client application is collectively known as the Rich Client Platform[13].

Applications other than IDEs can be built using a subset of the platform. These rich applications are still based on a dynamic plug-in model, and the UI is built using the same toolkits and extension points. The layout and function of the workbench is under fine-grained control of the plug-in developer in this case.

2.6 The Xpand Language

Xpand language was proposed by Open Architecture Ware (oAW) and is used for Model-to-Text (M2T) transformations. The language is offered as part of the Eclipse Modeling Project (EMP). The language allows the developer to define a set of templates that transform objects that exist in an instance of a model into text. Major advantages of Xpand are the fact that it is source model independent, which is usually source code but it can be whatever text the user desires, and its vocabulary is limited, allowing for a quick learning

curve. The language requires as input a model instance, the model and the transformation templates. Xpand first validates the instance through the provided model and then, as the name suggests, expands the objects found in the instance with the input templates. It allows the user to define, except form the expansion templates, functions implemented in Java language using the Xtext functionality. Xpand is a markup language and uses the "<<" and ">>" to mark the start and the end of the markup context. Enables code expansion using the model structure (i.e. expanding all child elements of a specific type inside a node) and supports if-then-else structure. Functions call be called inside markup. The advantages of Xpand are the fact that it is source model independent, its vocabulary is limited allowing for a quick learning curve while the integration with Xtend allows for handling complex requirements. Then, EMP allows for defining workflows that can help a modeler to achieve multiple parsings of the model with different goals.

2.7 Java Agent Development Framework

JADE (Java Agent Development Framework) is a software Framework fully implemented in the Java language. It simplifies the implementation of multi-agent systems through a middle-ware that complies with the FIPA specifications and through a set of graphical tools that support the debugging and deployment phases. A JADE-based system can be distributed across machines (which not even need to share the same OS) and the configuration can be controlled via a remote GUI. The configuration can be even changed at run-time by moving agents from one machine to another, as and when required. JADE is completely implemented in Java language and the minimal system requirement is the version 5 of JAVA (the runtime environment or the JDK).

Besides the agent abstraction, JADE provides a simple yet powerful task execution and composition model, peer to peer agent communication based on the asynchronous message passing paradigm, a yellow pages service supporting publish subscribe discovery mechanism and many other advanced features that facilitates the development of a distributed system.

2.8 The ASEME Methodology

The Agent Systems Engineering Methodology (ASEME) [15] is a recently emerging methodology for developing multi-agent systems. Its major advantages to existing methodologies are that it builds on existing languages such as statecharts and UML (which are familiar to engineers) in order to represent system analysis and design models. It provides three different levels of abstraction, thus catering for large-scale systems involving diverse technologies. It is agent architecture and agent mental model independent, allowing the designer to select the architecture type and the mental attributes of the agent that he prefers (e.g. procedural agents, belief-desire-intentions (BDI) agents, etc). Moreover, the ASEME process follows the modern model driven engineering style, thus the models of each phase are produced by applying transformation rules to the models of the previous phase. Each phase adds more detail and becomes more formal leading gradually to implementation. Thus, ASEME is a model-driven engineering process that can be automated by using rules for models transformation and knowledge for adding detail in every development phase. We define a platform independent model at the end of the design phase that describes the system and allows its implementation with the use of different platforms or programming languages. In this thesis we implemented an integrated development environment(IDE) guiding the user to follow the model transformation process for implementing a multi-agent system using the popular Java Agent Development Framework (JADE).

2.8.1 ASEME process overview

The software development phases of the Agent Systems Engineering Methodology (ASEME)[15] are presented in Figure 2.1. There are six phases, the first four produce system models (development phases), while the last two (verification and optimization phases) evaluate and optimize these models. The process is iterative allowing for incremental development and provides the original possibility to jump backwards to any previous phase due to the utilized model driven engineering (MDE) approach.

MDE is the systematic use of models as primary engineering artifacts throughout the engineering life-cycle. It is compatible with the Model Driven Architecture (MDA) paradigm of the Object Management Group (OMG). MDA's strong point is that it strives for portability, interoperability and

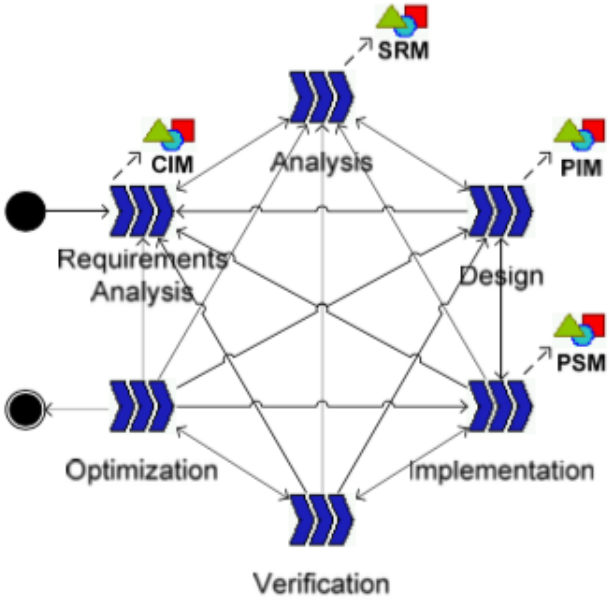


Figure 2.1: ASEME process overview ([15])

re-usability, three non-functional requirements that are very important for modern systems design. MDA defines three models:

- A computation independent model (CIM) is a view of a system that does not show structural details of systems. It uses a vocabulary that is familiar to the practitioners of the domain in question as it is used for system specification
- A platform independent model (PIM) is a view of a system that on one hand provides a specific technical specification of the system, but on the other hand exhibits a specified degree of platform independence so as to be suitable for use with a number of different platforms
- A platform specific model (PSM) is a view of a system combining the specifications in the PIM with the details that specify how that system uses a particular type of platform

In ASEME the CIM, PIM and PSM are the models outputted by the requirements analysis, design and implementation phases respectively. We

have inserted another model as the output of the analysis phase, the System Roles Model (SRM). Each of these models is produced by applying simple transformation rules to the previous phase model and this transformation is traceable, that is it can be reverse engineered.

We define three levels of abstraction in each phase. The first is the societal level. There, the whole multi-agent system functionality is modeled. Then, in the agent level, we model (zoom in) each part of the society, the agent. Finally, we focus in the details that compose each of the agent's parts in the capability level. We define the concept of capability as the ability of an agent to achieve specific tasks that require the use of one or more functionalities. The latter refer to the technical solution(s) to a given class of tasks. Moreover, capabilities are decomposed to simple activities, each of which corresponds to exactly one functionality. Thus, an activity corresponds to the instantiation of a specific technique for dealing with a particular task. ASEME is mainly concerned with the first two abstraction levels assuming that development in the capability level can be achieved using classical (or even technology-specific) software engineering techniques. In Figure 2.2, we present the ASEME phases, the different levels of abstraction and the models related to each one of them

2.8.2 The AMOLA metamodels

System Actor Goal model (SAG)

The SAG model[17] is a subset of the Actor model of the Tropos ecore model. Tropos is, on one hand, one of the very few AOSE methodologies that deal with requirements analysis, and, on the other hand it borrows successful practices from the general software engineering discipline. The Tropos diagrams provide more concepts than the ones used by AMOLA as they are also used for system analysis, however, AMOLA defines more well-suited diagrams for system analysis.Thus, the AMOLA System Actors Goals diagram is the one that appears in Figure 2.3 employing the Actor and Goal concepts. The actor references his goals using the EReference my goal, while the Goal references a unique depender and zero or more dependee(s). The reader should notice the choice to add the requirements EAttribute of Goal. Through this attribute, each goal is related to functional and non-functional requirements, which are documented in plain text form.

	Development Phase	Levels of Abstraction		
		Society Level	Agent Level	Capability Level
CIM	**Requirements Analysis** *AMOLA Models*	Actors Actor Diagram	Goals Actor Diagram	Requirements Requirements per goal
SRM	**Analysis** *AMOLA Models*	Roles and Protocols Use case Diagram, Agent Interaction Protocols	Capabilities Roles Model	Functionalities Functionality Table
PIM	**Design** *AMOLA Models*	Society Control Inter-agent control model, Ontology, Message Types	Agent Control and Modules Intra-agent control model	Components
PSM	**Implementation**	Platform management code	Agent code	Capabilities code
	Verification	Protocols testing	Agent testing	Component testing
	Optimization	Number of instantiated agents	Agent resources	Code optimization

Figure 2.2: ASEME phases and their products ([15])

Use case model (SUC)

In the analysis phase, the analyst needs to start capturing the functionality behind the system that is under development. In order to do that he needs to start thinking not in terms of goal but in terms of what will the system need to do and who are the involved actors in each activity. The use case diagram helps to visualize the system including its interaction with external entities, humans or other systems. It is well-known by software engineers as part of the Unified Modeling Language (UML). In AMOLA no new elements are needed other than those proposed by UML, however, the semantics change. Firstly, the actor "enters" the system and assumes a role. Agents are modeled as roles, either within the system box (for the agents that are to be developed) or outside the system box (for existing agents in the environment). Human actors are represented as roles outside the systembox (like in traditional UML use case diagrams). This approach aims to show the concept that we are modeling artificial agents interacting with other artificial agents or human agents. Secondly, the different use cases must be directly related to at least one artificial agent role. The SUC metamodel containing the concepts

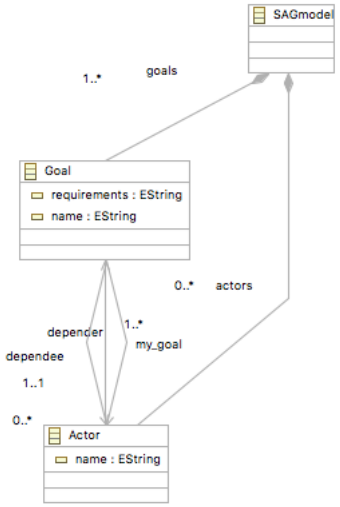


Figure 2.3: SAG emf meta-model

used by AMOLA is presented in Figure 2.4. The concept UseCase has been defined that can include and be included by other UseCase concepts. It interacts with one or more roles, which can be one of the types defined in RoleType enumeration :

- Abstract
- System
- Legacy System
- External System
- Human

AIP model

Another part of the analysis phase is the Agent Interaction Protocols(AIP). Protocols(in the society level) originate from use cases that connect two artificial agent roles. For defining protocols in the AIP metamodel (Figure 2.5) the concepts of Protocol and Participant are used with the condition that each protocol needs at least two participant roles.

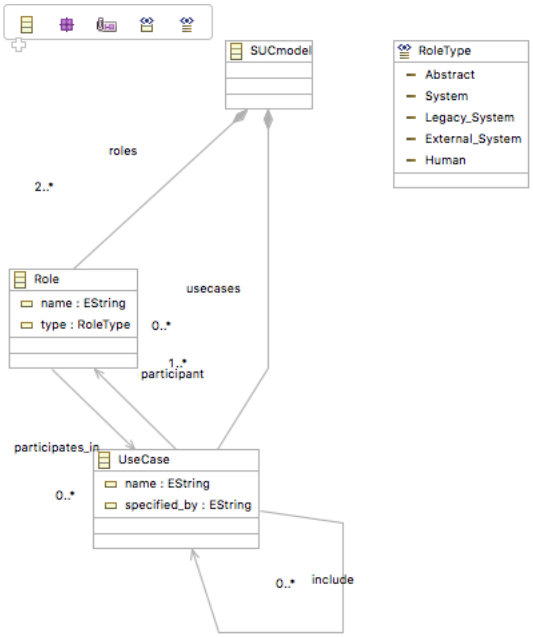


Figure 2.4: SUC emf meta-model

SRM model

An important concept in AOSE is the role. An agent is assumed to undertake one or many roles in his lifetime. The role is associated with activities and this is one of the main differences with traditional software engineering, the fact that the activity is no longer associated with the system, rather with the role. Moreover, after defining the capabilities of the agents and decomposing them to simple activities in the SUC model we need to define the dynamic composition of these activities by each role so that he achieves his goals. Thus, the SRM model, was defined, based on the Gaia Role model. Gaia defines the liveness formula operators that allow the composition of formulas depicting the role's dynamic behavior. However, we needed to change the role model of Gaia in order to accommodate the integration in an agent's role the incorporation of complex agent interaction protocols (within which an agent can assume more than one roles even at the same time), a weakness of the Gaia methodology. The AMOLA SRM metamodel is presented in Figure 2.6. The SRM metamodel defines the concept Role that references the concepts:

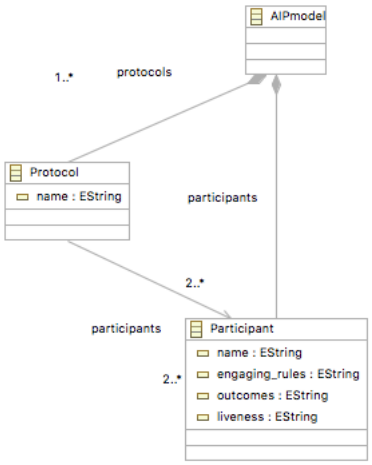


Figure 2.5: AIP emf meta-model

- Activity, that refers to a simple activity with one attribute, name (its name) and one reference, functionality (the description of what this activity does),
- Capability that refers to groups of activities (to which it refers) achieving a high level goal, with two attributes, name (its name) and description (its description)

The Role concept also has the name and liveness attributes (the first is the rolename and the second its liveness formula). The reader should also note the functionality concept referenced by the Activity concept which is used to associate the activity to a generic functionality. Functionality has the following attributes :

- description, the functionality description
- algorithm, the algorithm used by the functionality
- permissions, the needed permissions
- technology, the used technology
- environment, the proper environment

all in free text form.

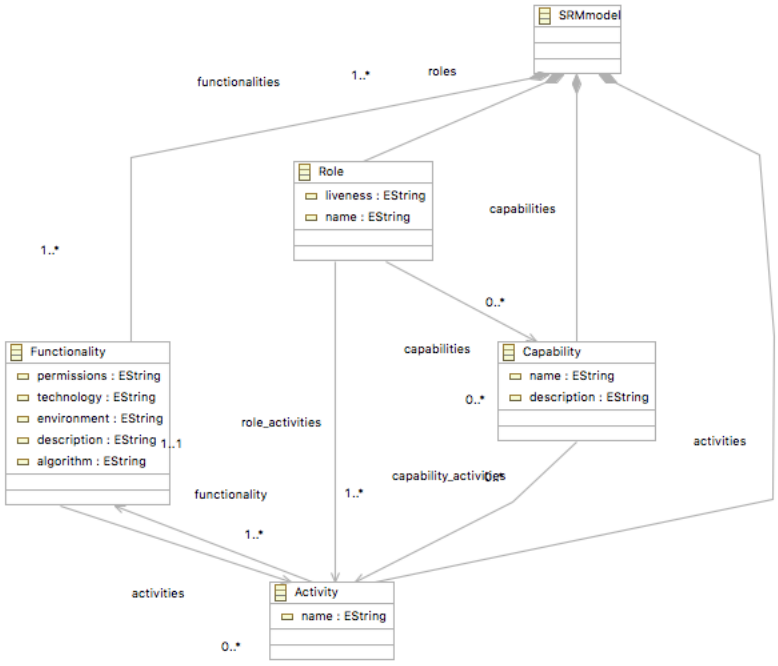


Figure 2.6: SRM emf meta-model

EAC/IAC model

In order to represent system designs, AMOLA is based on statecharts, a well-known and general language and does not make any assumptions on the ontology, communication model, reasoning process or the mental attitudes (e.g. belief-desire intentions) of the agents, giving this freedom to the designer. Other methodologies impose (like Prometheus or INGENIAS [2]), or strongly imply (like Tropos [2]) the agent mental models. Of course, there are some developers who want to have all these things ready for them, but there are others who want to use different agent paradigms according to their expertise. For example, one can use AMOLA for defining Belief Desire-Intentions based agents, while another for defining procedural agents. The inspiration for defining the IAC metamodel mainly came from the UML statechart definition. Aiming to define the statechart using the AMOLA definition of statechart, the IAC metamodel differs significantly from the

UML statechart. However, a UML statechart can be transformed to an IAC statechart although some elements would be difficult to define (UML does not cater for transition expressions and association of variables to nodes and uses statecharts to define a single object's behaviour). Thus, the IAC metamodel, which is presented in Figure 2.7, defines a Model concept that has nodes, transitions and variables EReferences. Note that it also has a name EAttribute. The latter is used to define the namespace of the IAC model. The namespace should follow the Java or C"#" modern package namespace format. The nodes contain the following attributes:

- name. The name of the node
- type. The type of the node, corresponding to the type of state in a statechart, typically one of AND, OR, BASIC, START, END, CONDITION
- label. The node's label
- activity. The activity related to the node

Nodes also refer to variables. The Variable EClass has the attributes name and type(e.g. the variable with name "count" has type "integer"). The next concept defined in this metamodel is that of Transition, which has four attributes:

- name, usually in the form <source node label>TO<target node label>
- TE, the transition expression. This expression contains the conditions and events that make the transition possible. Through the transition expressions (TEs) the modeler defines the control information in the IAC. TEs can use concepts from an ontology as variables. Moreover, the receipt or transmission of an inter-agent message can be used (in the case of agent interaction protocols).
- source, the source node
- target, the target node

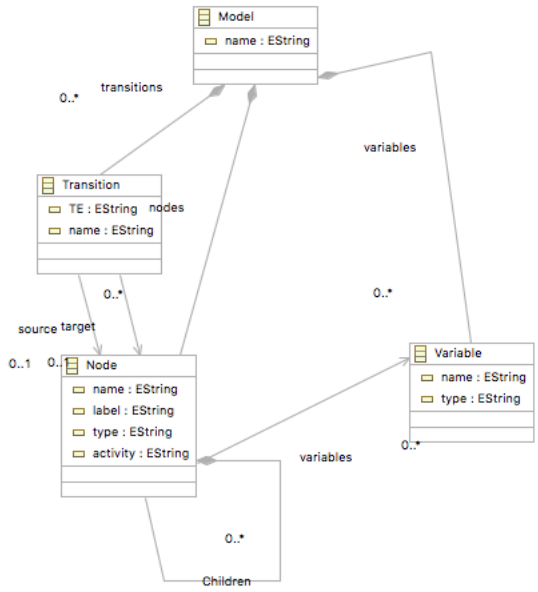


Figure 2.7: Statechart emf meta-model

2.8.3 The ASEME Model-Driven Process

1. Edit SAG model. The business consultant of the software development firm identifies the actors involved in the system to be along with their goals.

2. SAG2SUC. An automated task, as the reader can see in the figure this task has only a mandatory input model (SAG) and an output model (SUC). It creates an initial SUC model based on the previously created SAG model.

3. Refine Use Cases. The analyst works on the SUC model and refines the general use cases using the include relationship. He/she also identifies which actors will be implemented defining them as human or artificial agent actors. The overall system design is enriched by identifying the tasks that have to be carried out by the actors.

4. SUC2SRM. An automated task, it has only a mandatory input model (SUC) and an output model (SRM). It creates an initial SRM model

based on the previously created SUC model.

5. SRM2IAC. An automated task, it has only a mandatory input model (SRM) and an output model (IAC). It creates multiple initial IAC models based on the previously created SRM model, one for each role.

6. Refine the IAC model. The designer works on each IAC model by defining the conditions and/or events that will enable the transitions from one task to the other.

7. IAC2JADE. An automated task, it has only a mandatory input model (IAC) and an output model (Java JADE Agent and Behaviours code). It creates a JADE Agent class and multiple JADE Behaviour classes for each IAC model.

8. Write SimpleBehaviour action methods. The programmer writes code only for the JADE SimpleBehaviour class descendants' action methods.

2.9 BPMN/XPDL

2.9.1 BPMN

Business Process Model and Notation (BPMN) [8] is a standard for business process modeling that provides a graphical notation for specifying business processes in a Business Process Diagram (BPD), based on a flowcharting technique very similar to activity diagrams from Unified Modeling Language (UML) 5 . The objective of BPMN is to support business process management, for both technical users and business users, by providing a notation that is intuitive to business users, yet able to represent complex process semantics. The BPMN specification also provides a mapping between the graphics of the notation and the underlying constructs of execution languages. The primary goal of BPMN is to provide a standard notation readily understandable by all business stakeholders. These include the business analysts who create and refine the processes, the technical developers responsible for implementing them, and the business managers who monitor and manage them. Consequently, BPMN serves as a common language, bridging the communication gap that frequently occurs between business process design and implementation. BPMN is constrained to support only the concepts of modeling applicable to business processes. In addition, while BPMN shows

the flow of data, and the association of data artifacts to activities, it is not a data flow diagram.

2.9.2 XPDL

The XML Process Definition Language (XPDL) is a format standardized by the Workflow Management Coalition (WfMC) to interchange business process definitions between different workflow products, i.e. between different modeling tools and management suites. XPDL defines an XML schema for specifying the declarative part of workflow / business process.

XPDL is designed to exchange the process definition, both the graphics and the semantics of a workflow business process. XPDL is currently the best file format for exchange of BPMN diagrams; it has been designed specifically to store all aspects of a BPMN diagram. XPDL contains elements to hold graphical information, such as the X and Y position of the nodes, as well as executable aspects which would be used to run a process. This distinguishes XPDL from BPEL which focuses exclusively on the executable aspects of the process. BPEL does not contain elements to represent the graphical aspects of a process diagram. It is possible to say that XPDL is the XML Serialization of BPMN.

Chapter 3

ASEME IDE Implementation

3.1 Dashboard

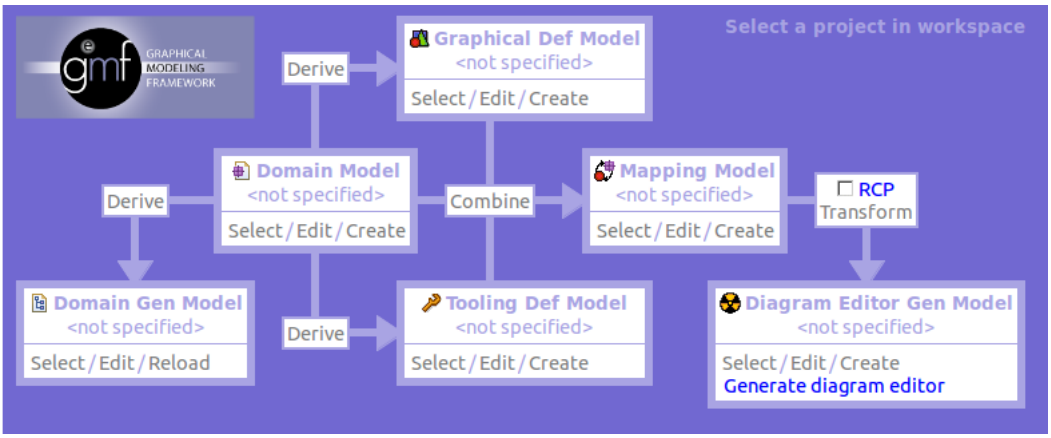
Creating an IDE for ASEME had a certain challenge : how would a user who is not familiar with the methodology and/or the concepts would cope with an admittedly pretentious process such as ASEME. Our approach was to use a Dashboard.

3.1.1 Dashboard layout

Following the paradigm of Eclipse GMF Dashboard (Figure 3.1) we created the ASEME Dashboard view (Figure 3.2) to help the user work through the flow of the ASEME process. As can be observed, it invokes actions for many of the steps one routinely uses during the development of a multi-agent system (MAS) using the ASEME methodology, but all from a single location, gathering together the presentation of the model-driven process and showing the discrete development steps, as well as the model transformations between the different development phases. On both figures the dashboards are presented at their default state, empty, with "not specified" as the default text for the empty model file label.

For the ASEME Dashboard layout we kept the same layout principles followed by GMF Dashboard. Rectangular shapes for our models with the available actions to the bottom and using arrows to visualize the flow and guide the user through the Model-Driven process. We also used dashed arrows to imply the optional parts of it.

Figure 3.1: GMF Dashboard



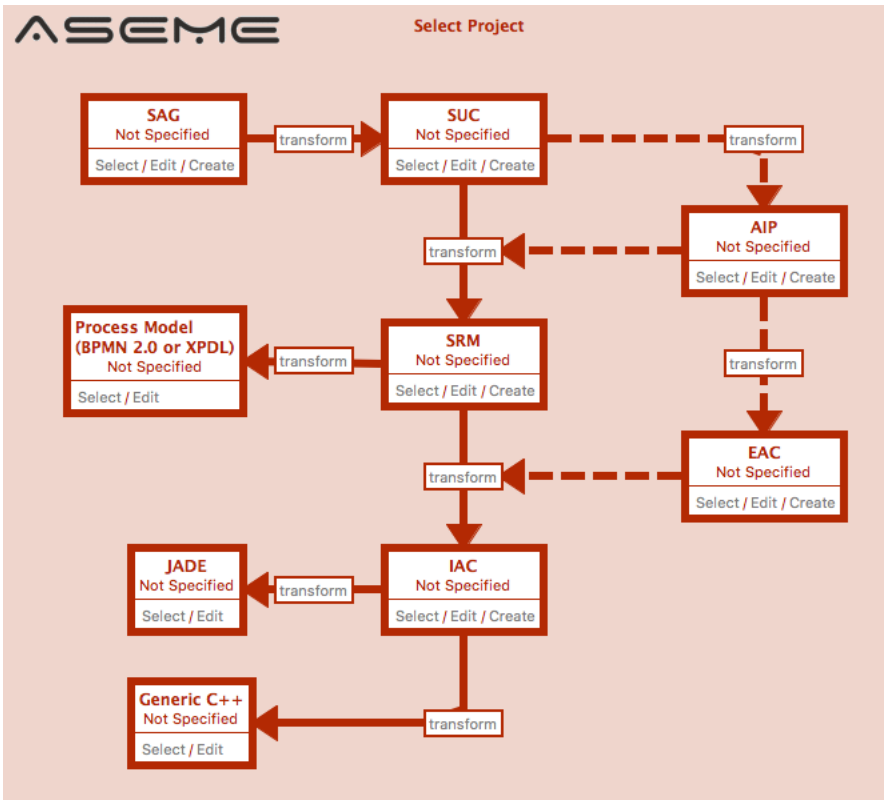


Figure 3.2: ASEME Dashboard

3.1.2 Dashboard architecture

ASEMEDashboardView plugin contains three packages :

1. asemedashboardview that contains only the Activator class. An activator is a Java class that controls the plug-in's life cycle.
2. asemedashboardview.views that contains the main code for the implementation of the dashboard
3. asemedashboardviews.actions that contains the implementation of each of the transform actions on the dashboard

In the Eclipse Platform a view is typically used to navigate a hierarchy of information, open an editor, or display properties for the active editor. ASEME Dashboard is implemented as a view, thus the ASEMEDashboardView plugin project has a view extension added to the plugin.xml file and the ASEMEDashboardView.java is the name of the required org.eclipse.ui.parts.ViewPart subclass, located in the .views package. In the same package there are the three interfaces used to provide the functionality of the dashboard.

- ASEMEFacade : contains the initial API
- ASEMEAction : allows other plugins to contribute actions to the dashboard
- ActionContainer : API for add/remove actions

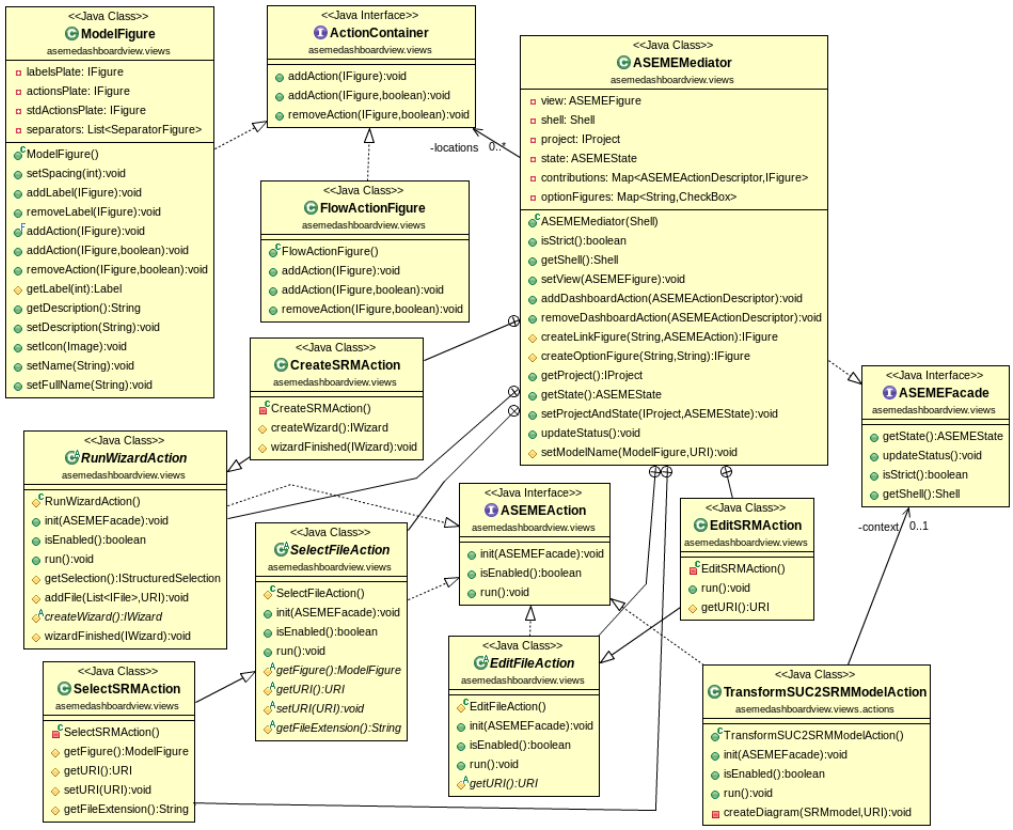


Figure 3.3: ASEME Dashboard class diagram

In figure 3.3 you can see a class diagram of the ASEME Dashboard.

The implementation class for ASEMEFacade is ASEMEMediator. ASEMEMediator class is mandatory for the dashboard. From the setView method the actions for the models (Create, Select, Edit) can be registered/unregistered.

RunWizardAction, SelectFileAction and EditFileAction are the Abstract classes for each Create, Select and Edit Actions shown on the dashboard and all three implement ASEMEAction interface. All they above are defined as inner classes of ASEMEMediator. Note that only the actions concerning SRM model(Create, Edit, Select, Transform) are shown in the class diagram mainly for readability reasons.

Transform actions also implement ASEMEAction interface and have any context needed through a private ASEMEFacade type variable named context (see TrasformSUC2SRMModelAction in Figure 3.3)

The Java class of the implementation of the dashboard layout is ASEMEFigure.java in asemedsahboardview.views package and extends org.eclipse.draw2d.RectangleFigure. We mostly used org.eclipse.draw2d and org.eclipse.draw2d.geometry packages. For the layout we used ModelFigure and FlowActionFigure that implement ActionContainer interface(also shown in class diagram) and visually represent the Model boxes and the transform boxes respectively. More detailed technical description on customizing the dashboard is presented in the next chapter.

Aside from the classes discussed until now we have a registry for the actions (ASEMEActionRegistry), a class holding the state(ASEMEState) and some helper classes.

3.1.3 Benefits

A dashboard is an easily readable, one page summary of the analysis of the information. It is an overview of a system at a glance, thus there are benefits that result in the utilization of such a tool :

Dashboards could be customized in terms of users and expectations. A dashboard can be customized to present useful information and also can be extended in terms of functionality . This allows each user to see the level of detail that they need in order to meet their goals and the developer the possibility to easily make functionality additions and modifications. In the past users would spend large amount of time reviewing and analyzing different reports to end in a final conclusion. This tool allows to see, at a glance, an overall situation report of the desired information. But, having all-in-one does not means the absence of details. Dashboards are developed with the ability to get as deeper in information as required by simply selecting the desired variable or object. More specifically the use of ASEME Dashboard reduces significantly the time needed for designing a MAS with ASEME in comparison with the separate use of the preexisting tools while the same time gives the user an overview of the process. Finally there is no need for complicated and exhaustive training. Dashboards are design to be intuitive to any user. The graphic design allows an easy and smooth navigation throughout the information.

The dashboard approach provides a significant opportunity to make ASEME designing more efficient and quick to learn.

3.2 Metamodels and Editors

In this section we will present the changes and additions made to the existing ASEME metamodels and editors. Each of the AMOLA metamodels is implemented as a EMF model and has its own GMF editor generated and customized. So for the better understanding of the reader considered appropriate to mention some general information about their EMF/GMF implementation and structure, and describe some general procedures followed for editing emf models and gmf editors before getting in a more detailed description of the specific changes made.

GMF uses six files to create a generated graphical editor for instances of a given metamodel :

- **Domain Model** : the metamodel we want to use to create the graphical editor. For this metamodel.There is a choice between several kinds of metamodels : Annotated Java code, Ecore model, Rose class model, UML model or XML Schema). For this thesis impementation we used Ecore models **(.ecore)**.
- **Domain Generation Model (.genmodel)** : this file is used to generate the domain model code with EMF (it is the EMF file genModel)
- **Graphical Definition Model (.gmfgraph)** : this file is used to define the graphical elements for the domain model
- **Tooling Definition Model (.gmftool)** : this file is used to define the palette of tools that can be used in the graphical editor
- **Mapping Model (.gmfmap)** : this file links the domain model, the graphical model (.gmfgraph) and the tooling model (.gmftool)
- **Diagram Editor Generation Model (.gmfgen)** : this final file is used to generate the GMF graphical editor in addition to the EMF code generated by the .genmodel file

Starting with the ASEME IDE structure, these six files mentioned above for each of the metamodels are located in a Eclipse plugin project named as MetamodelNameDesign (SAGDesign, SUCDesign etc.) in the "model" folder. The first two files (.ecore, .genmodel) is the EMF implementation of the given model.From the .genmodel file you can generate Model, Edit and

Editor code. The Model code (Java implementation of the model) is located inside the same plugin in the "src" folder, the Edit (Java code for editing of model objects) and Editor code (the UI for the EMF editor of the model objects and wizard) are located in generated plugin projects with the .edit and .editor suffix respectively. There is also a .diagram project (the GMF editor) generated from the .gmfgen file.

In order to modify the metamodels implementation and apply the changes the following steps are needed :

1. Edit the .ecore file from the default editor provided by EMF framework and save it
2. Reload the .ecore model to the .genmodel file and
3. Re-generate the java code in order for the changes to take effect (from .genmodel)

For the GMF editors modifications the re-generation step is the same(from the .gmfgen file) , but the editing takes part in .gmftool, gmfgraph and .gmfmap files and according to changes some steps of the GMF editor design process (Figure 3.1) might need to be repeated. Further details for modifying GMF editors will be presented later in this chapter.

For the readers convenience we present two figures of the generated editors, as called throughout this thesis the EMF Editor(Figure 3.4) and the GMF Editor(Figure 3.5)

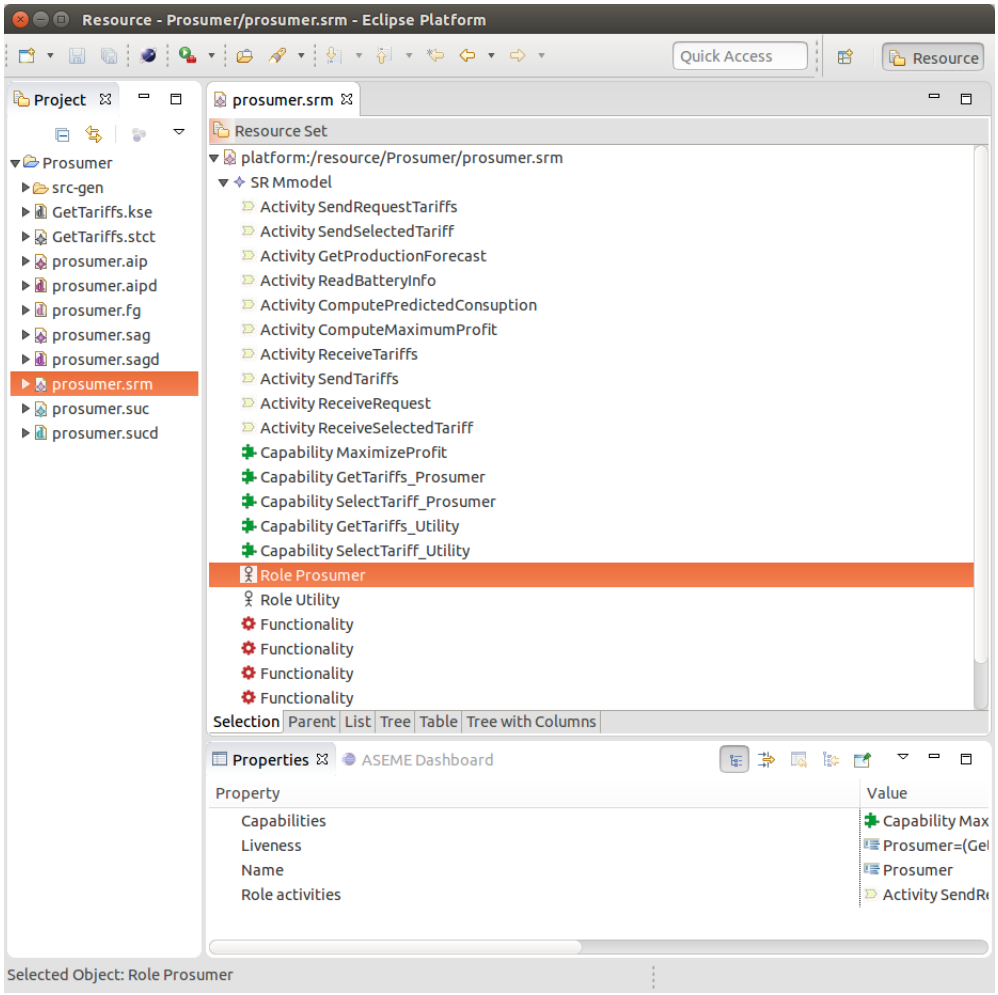


Figure 3.4: SRM meta-model EMF Editor

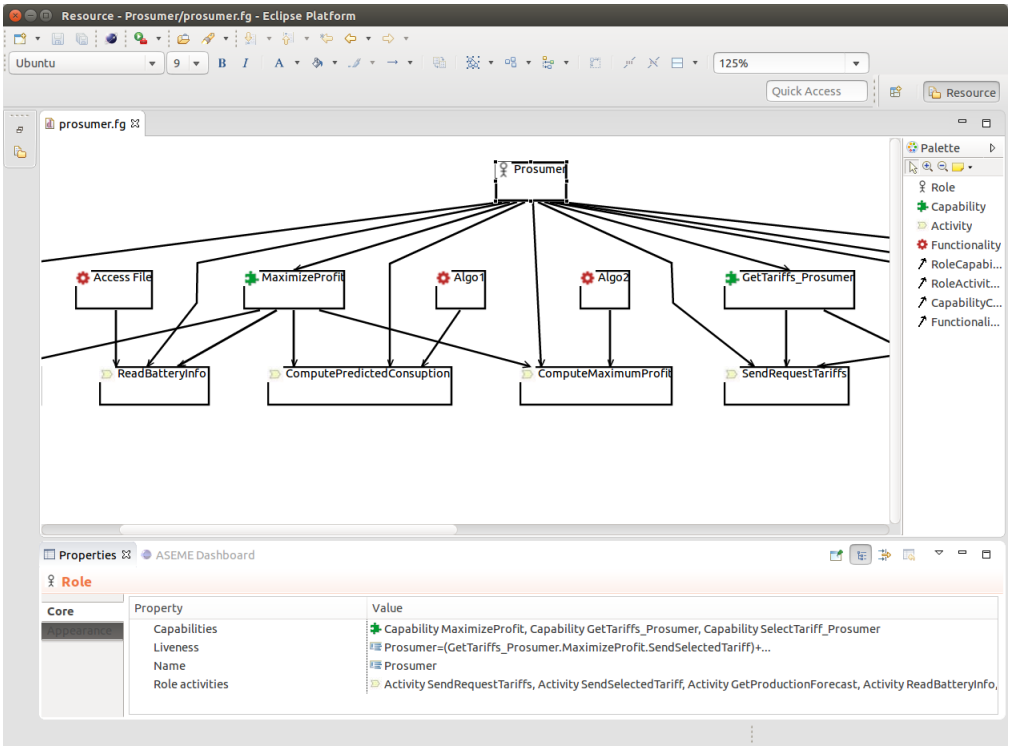


Figure 3.5: SRM meta-model GMF Editor

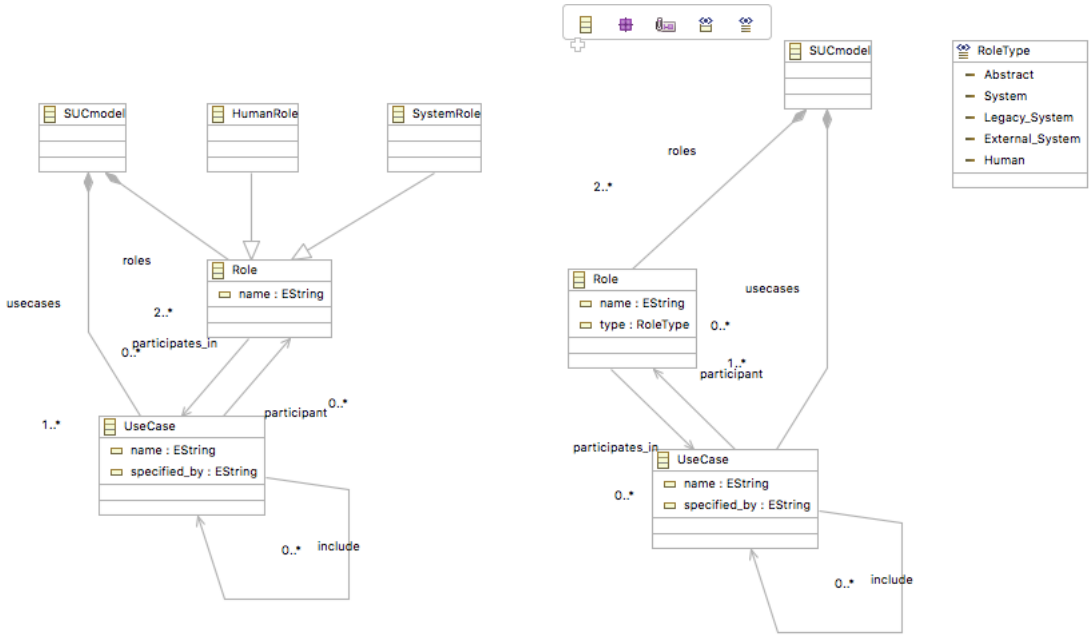
At the EMF level, SAG and Statechart metamodels remained unchanged, so a more detailed description of the changes to the rest of the models follows.

3.2.1 SUC

In SUC meta model we removed the HumanRole and SystemRole EObjects, replacing the way of describing a Role on the SUC level with the enumeration RoleType Abstract, System, `Legacy_System, External_System`, Human and adding a RoleType field/EAttribute to the Role EObject (fig. 3.6).

3.2.2 AIP

In AIP metamodel we made only one change at this level : the `engaging_rules`, outcomes and liveness attributes of the Participant EObject changed to be multi-line Strings (editable from the properties view of each attribute at .genmodel file).



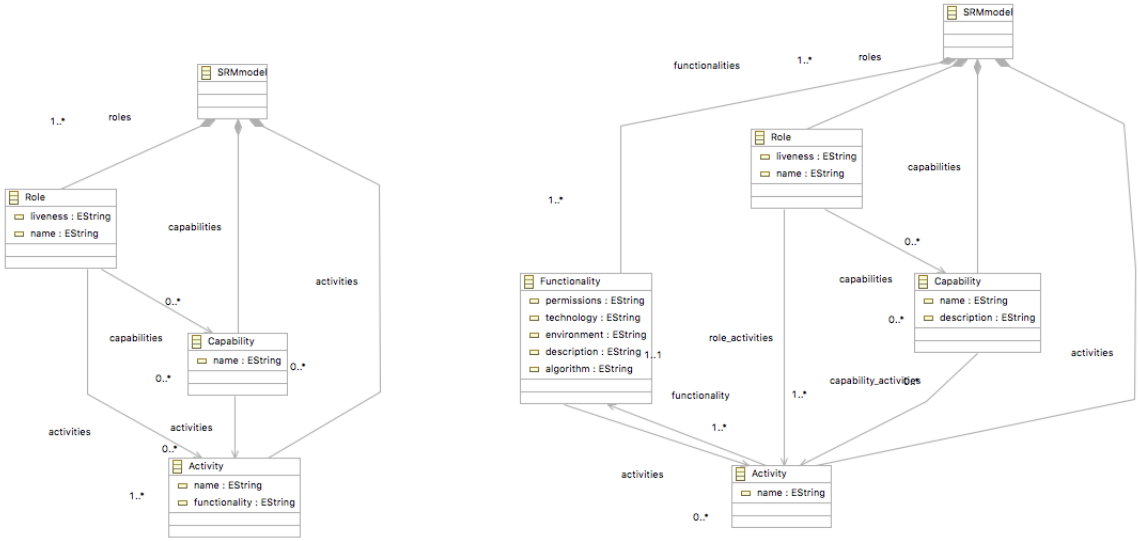
(a) SUC model before (b) SUC model refined

Figure 3.6: The evolution of SUC metamodel implementation

3.2.3 SRM

At the SRM metamodel we added an EString attribute to capability EObject named description, that obviously is the capabilitys' description in free text. Then we replaced the functionality EAttribite of Activity EObject with the new Functionality EObject containing five(5) EAttributes (all EStrings):

- description : the functionality description in free text
- technology : the technology this functionality uses
- environment : the environment
- permissions : the needed permissions
- algorithm : the algorithm



(a) SRM model before

(b) SRM model refined

Figure 3.7: The evolution of SRM metamodel implementation

3.2.4 Editors modifications

After editing the metamodels, the next step is to setup their GMF editors. The editing process varies depending to the part of the editor that needs to be modified, ie to change the palette we had to modify the .gmftool file then repeat the remaining process of creating the editor (regenerate .gmfmap and .gmfgen files and then regenerate the GMF editor code). As the best way to describe the editing process is by example, below we are going to see some variations by presenting a set of processes followed to do specific modifications to more than one GMF editors included in the ASEME IDE and then describe further customizations done to the Java code for each editor.

The framework provides the option to set the file extensions for the input files of the editor. The need to use this feature was obvious at some point, ie the SRM GMF editor designed to represent the functionality graph so a file extension like .fg would be way more suitable than the default `.srm_diagram`. This change can be done by editing the "Diagram File Extension" property of the Gen Editor Generator Name.diagram resource of the Name.gmfgen file as shown in figure 3.8 (for the SRM model). The file extensions used are .sagd, .sucd, aipd, .fg and .kse for SAG, SUC, AIP, SRM and Statechart respectively.

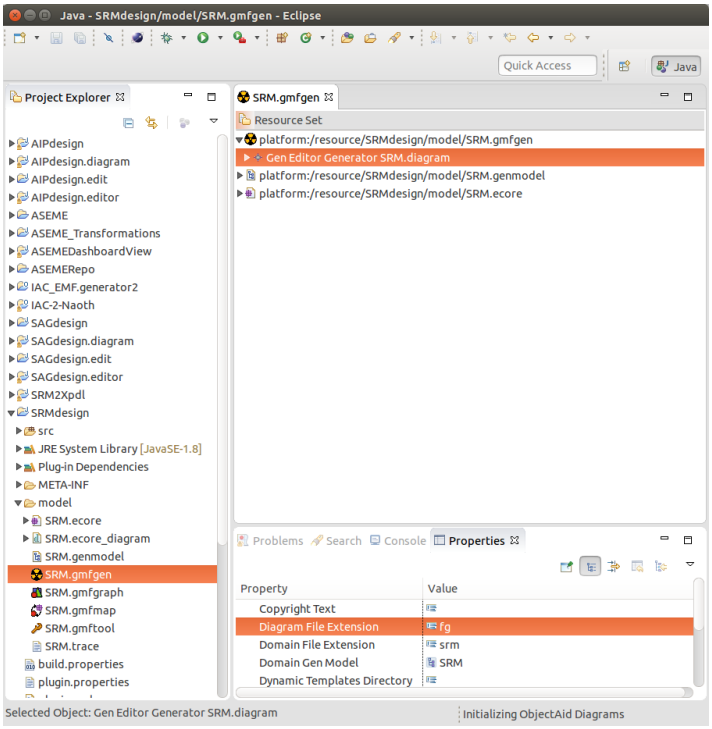


Figure 3.8: Change SRM diagram file extension to .fg

When the editor is generated, several icons are created in the plugin icons folders. Changing the default icons is mainly to replace these images. GMF uses Small(16x16) and large(32x32) .gif images, thus as long as we have the new images (and resize them if needed) we replace (which means delete the existing .gif file, copy/paste the .gif file and rename it with the deleted file name) the files at specific locations according to our needs :

Diagram editor icons modification

for the SRM model we replaced these files:

- /SRM.diagram/icons/obj16/SRMDiagramFile.gif
- /SRM.edit/icons/full/obj16/Activity.gif

- /SRM.edit/icons/full/obj16/Capability.gif
- /SRM.edit/icons/full/obj16/Role.gif
- /SRM.edit/icons/full/obj16/Functionality.gif

As you can see at the example above the names of the files are quite descriptive and as obj16 folder implies all our new files must be 16x16 pixels images.

Tools palette icons modification

If you don't modify the tools palette icons, GMF will select the default EMF icons (the icons that are located in icons folder of the .edit project), which most of the times are not very representative. It is possible to override this and we used this feature for the SUC and the SRM diagram editors.

For the following example we will use the process followed for the SRM model. Since our purpose here is to show the process we assume we have the new icons we want to use for the SRM palette named Activity16x16.gif, Capability16x16.gif, Role16x16.gif and Functionality16x16.gif and Activity32x32.gif, Capability32x32.gif, Role32x32.gif and Functionality32x32.gif (the selected images and the selection criteria images will be presented to the corresponding editors subsections).

- In SRMdesign.edit/icons/full/obj16, replace Activity.gif, Capability.gif, Role.gif and Functionality.gif with the 16x16 images.
- In SRMdesign.edit, create a folder icons/full/obj32
- Copy the 32x32 icons
- Rename these files into Activity.gif, Capability.gif, Role.gif and Functionality.gif
- Open the SRM.gmftool file
- Under the Creation Tool Role :
  - delete both Default image nodes
  - Add a small icon bundle image with these properties

        * Bundle : SRMdesign.edit (plugin name in which the icons are located)
        * Path : icons/full/obj16/Role.gif (plugin relative path of the icon)
    - Add a large icon bundle image accordingly :
        * Bundle : SRMdesign.edit (plugin name in which the icons are located)
        * Path : icons/full/obj32/Role.gif (plugin relative path of the icon)

- Repeat the same operation under the creation tools for Activity, Capability and Functionality. The final form of the .gmftool file is shown in figure 3.9
- Click on the Transform label of the dashboard
- Regenerate the diagram editor

Most of the times the icon order on the palette is different than the order of the creation tools in the .gmftool file. In SUC and SRM GMF editors we changed the order of the palette icons by modifying the generated Java code. The Java method that must be modified is createSRM1Group() located in SRMPaletteFactory.java in SRM.diagram.part package of the .diagram plugin (accordingly for SUC). Below we present the createSRM1Group() method after the modifications.

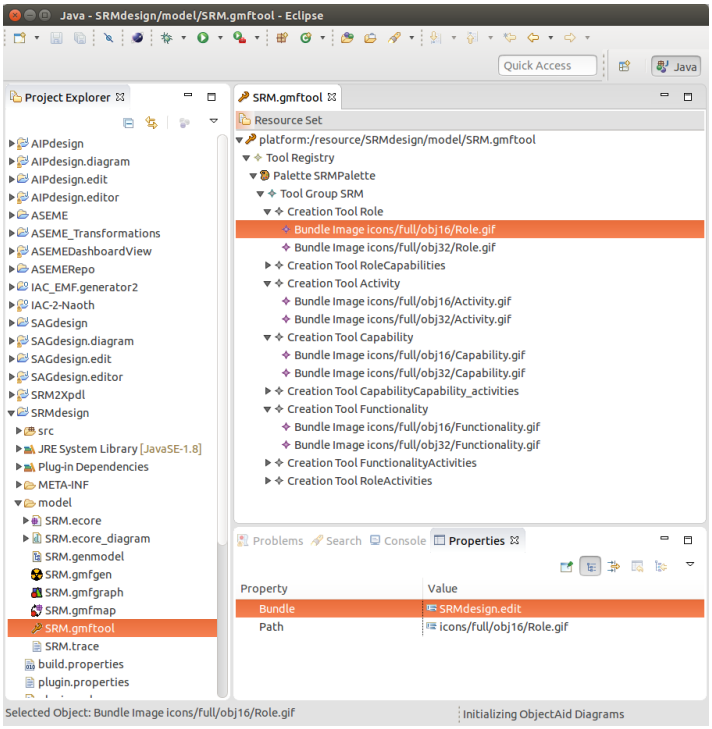


Figure 3.9: Final form of SRM.gmftool

Listing 3.1: Edited createSRM1Group()

```
/**
* Creates "SRM" palette tool group
* @generated NOT
*/
private PaletteContainer createSRM1Group() {
    PaletteGroup paletteContainer = new PaletteGroup(Messages.
        SRM1Group_title);
    paletteContainer.setId("createSRM1Group"); //$NON-NLS-1$
    paletteContainer.setDescription(Messages.SRM1Group_desc);
    paletteContainer.add(createRole1CreationTool());
    paletteContainer.add(createCapability4CreationTool());
    paletteContainer.add(createActivity3CreationTool());
    paletteContainer.add(createFunctionality6CreationTool());
    paletteContainer.add(createRoleCapabilities2CreationTool());
```

```
    paletteContainer.add(createRoleActivities8CreationTool());
    paletteContainer.add(
        createCapabilityCapability_activities5CreationTool());
    paletteContainer.add(createFunctionalityActivities7CreationTool());

    return paletteContainer;
}
```

As you can see we just changed the order of adding the creation tools to the paletteContainer Object. Also we added NOT after the generated annotation, which is used to state to the framework that this part of the code is modified by hand and avoid overriding it in future code re-generations of the code.

Another change made to all the GMF editors was to set a global line width and color for all the Diagram editor elements for the sake of a unified appearance, as they are all part of thee same tool. After a lot of testing during the development process we decided to set the global line width to 2 pixels and black as the default color. In order to do that we must edit the .gmfgraph file. Usually the first node under every Figure Descriptor is the selected Draw2d element (ie Rectangle, Ellipse for EObjects and Polyline Connection for links). Each of this elements has a Line Width property editable by the properties view and the color is added as a child in the tree structure under the nodes. In figure 3.10 you can see an example of SRM.gmfgraph file after the operation described for the RoleFigure. After doing this for all needed elements the .gmfmap and .gmfgen file must be regenerated and regenerate the editor (for each model) for the changes to take effect.

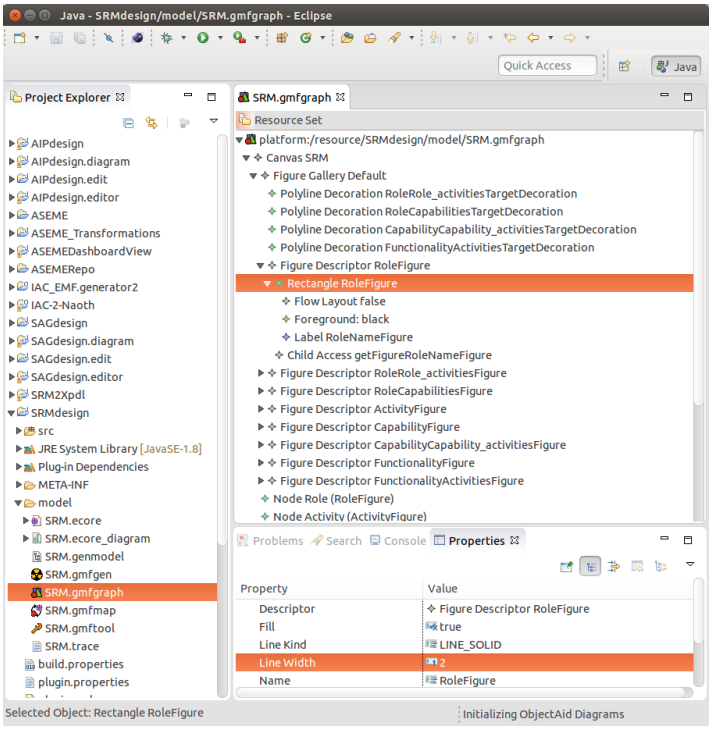


Figure 3.10: Set line width and color in SRM.RoleFigure

GMF also provides the possibility to use Audits for an editor. Audits represent set of rules that must hold true if evaluated on a diagram instance. Individual audits are represented by AuditRule and can be organized hierar-

chicaly into logical categories by using AuditContainer. To work with Audits you have to work on the .gmfmap file and then regenerate the editor as usual. We will present more detailed information about audits at SRM editor and Statechart editor subsections since we used audit rules one these two editors of the ASEME IDE.

So far we have covered all four basic variations of the editing process of a GMF editor since each one is starting from a different file (.gmfgen, .gmftool, .gmfgraph, .gmfmap) and we will continue with further customizations per editor. In SAG editor we just set the global line width and color, so no further info is required.

3.2.5 SUC Editor

In SUC editor the role icon was changed with the UML role icon in order to help any user of the ASEME IDE conceptually, because the UML role is the most dominant role concept among the engineering field. We also added an ellipse as the usecase icon, also the most dominant visual representation of a usecase in the field. In addition for better visuals we changed the position of the UseCase.Name Label inside the UseCase figure. to do this we had to edit the .gmfgraph file and add a Center Layout child node to the UseCase ellipse figure and regenerate the editor.

Finally since a UseCase can include other UseCases we chose to visualize this with a Label with fixed `<<include>>` text on it on the link connecting the included UseCases. For the placement of this label at the middle of each connection we used org.eclipse.draw2d.MidpointLocator to the createContents() method of UseCaseIncludeEditPart.java in SUC.diagram.edit.parts package.

Listing 3.2: SUC createContens()

```
/**
* @generated NOT
*/
private void createContents() {
    fFigureIncludeLabel = new WrappingLabel();
    FigureIncludeLabel.setText("<<includes>>");
    fFigureIncludeLabel.setFont(FFIGUREINCLUDELABEL_FONT);
    this.add(fFigureIncludeLabel, new MidpointLocator(this, 0));
```

}

3.2.6 AIP Editor modifications

In AIP editor we made two changes to the files. First to the .gmfgraph file we changed the layout of the participant figure to vertical. To do this we set the property Vertical, of the FlowLayout node under the ParticipantFigure to true. We also changed the .gmfmap file in order to do read only the multi-line attributes of the Participant figure. For this there is a boolean Read Only attribute uder each Feature Label Mapping node in the Misc section (Figure 3.11). After doing this and regenerated the editor we added a tooltip to inform the ASEME IDE user that these attributes are editable via the Properties view. So we had to edit creteNodeFigure() method located in AIP.diagram.edit.parts.ParticipantEditPart.java

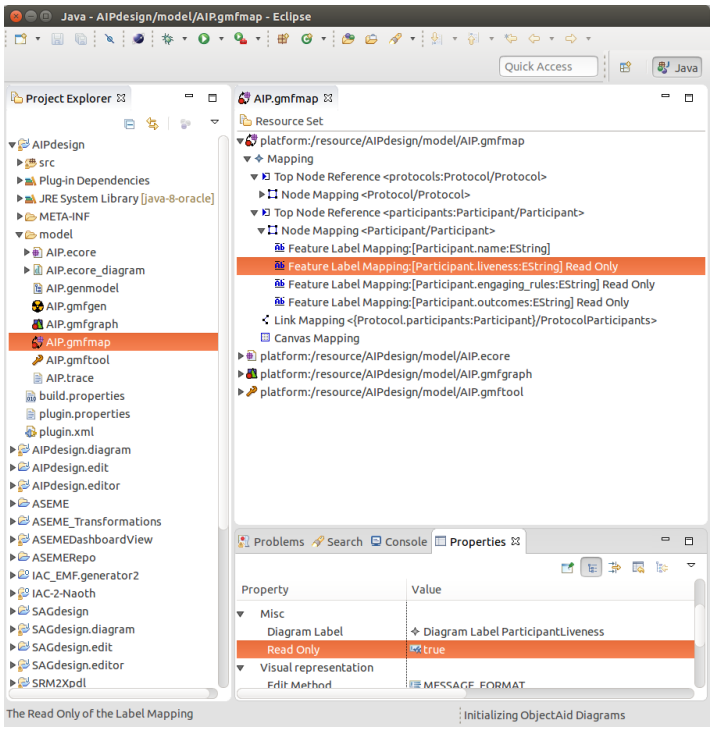


Figure 3.11: AIP.gmfmap set property to Read Only

Listing 3.3: Edited creteNodeFigure() for AIP Participant

```
/**
* Creates figure for this edit part.
* @generated NOT
*/
protected NodeFigure createNodeFigure() {
    NodeFigure figure = createNodePlate();
```

```
        Label tooltip = new Label("You can edit multiline attributes
            from the properties view ");
        figure.setToolTip(tooltip);
            figure.setLayoutManager(new StackLayout());
        IFigure shape = createNodeShape();
        figure.add(shape);
        contentPane = setupContentPane(shape);
        return figure;
    }
```

3.2.7 SRM Editor

After the refinements made to the SRM.ecore, we re-designed the SRM GMF editor to be the functionality graph of the SRM model. The icons for the palette selected with similar criteria as in the SUC editor, namely popular images for the representing concepts so we ended up in UML Role for SRM.Role, SPEM notation Activity for SRM.Activity, a jigsaw piece for SRM.Capability and a gear for SRM.Functionality.

After following the steps of the GMF Dashboard and generating our editor, we did the changes to the icons and the lines as described in Editors Modifications subsection we wanted to implement an automated process that adds the formula of a Capability as the last line of the liveness formula of the Role, when added and its not included at the Roles

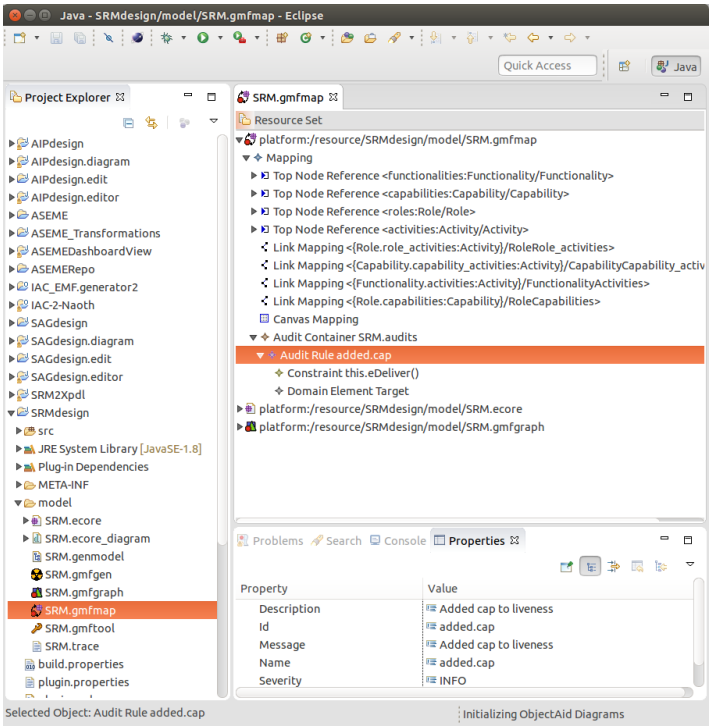


Figure 3.12: SRM.gmfmap Audit addition

Capabilities. To implement this (the functionality is similar to a listener) we had to add an Audit rule to the .gmfmap file as shown in (Figure 3.12), regenerate the editor and add the actual Java implementation of the desired functionality at the validate method of Adapter1 inner class located in SRM.diagram.providers.SRMValidationProvider.java.

Listing 3.4: SRM audit Java code

```
 /**
* @generated NOT
*/
public static class Adapter1 extends AbstractModelConstraint {

    /**
    * @generated NOT
    */
    public IStatus validate(IValidationContext ctx) {
        Role context = (Role) ctx.getTarget();

        String formula = context.getLiveness().replaceAll(" ", "");

        ArrayList<String> left = new ArrayList<String>();
        ArrayList<String> right = new ArrayList<String>();


        Pattern testPattern = Pattern.compile("\\w+_*w*");

        Matcher testMatcher = testPattern.matcher(formula);


        StringTokenizer line = new StringTokenizer(formula, "\n");

        while (line.hasMoreTokens()) {
            String tmp = line.nextToken();
            StringTokenizer formulaElements = new StringTokenizer(tmp,
                    "=");
```

```
    left .add(formulaElements.nextToken());
    right .add(formulaElements.nextToken());
}

boolean added = false;

for( Iterator<Capability> capIter = context.getCapabilities().
     iterator (); capIter.hasNext();){

    Capability tmpCap = capIter.next();

    boolean found = false;

    for (int i=1; i<left.size(); i++){

        if (tmpCap.getName().equals(left.get(i))){

            found = true;
            break;
        }

    }

    if (!found){

        String capForm = new String();

        if (tmpCap.getDescription()!=null){
            capForm = tmpCap.getName() + "=" + tmpCap.
                getDescription();
        }

        else{
            capForm = tmpCap.getName();
        }

        if (context.getLiveness().endsWith("\n")){
```

```
                context.setLiveness( context.getLiveness() + capForm+
                    "\n" );
            }
            else{
                context.setLiveness( context.getLiveness() +"\n"+
                    capForm+ "\n" );
            }

            added = true;

        }

    }


    return ctx.createSuccessStatus();

  }
}
```

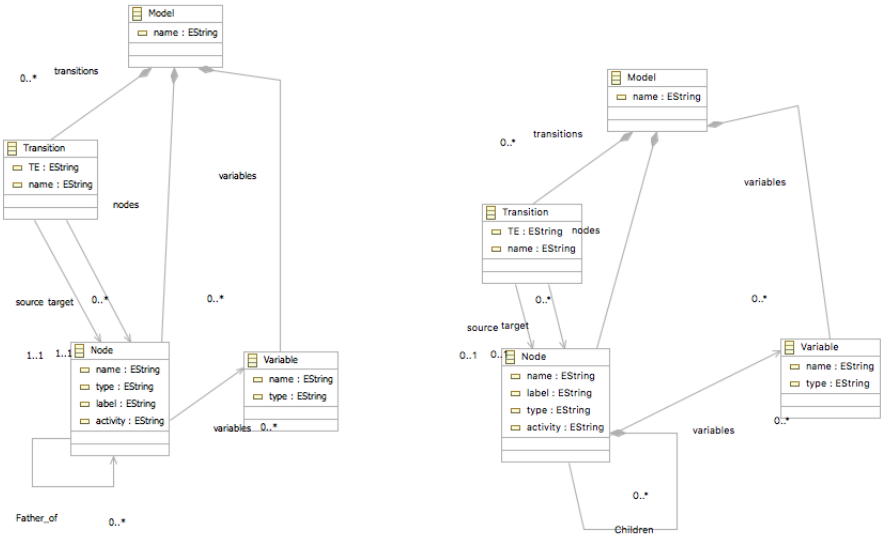
3.2.8 Statechart Editor

For the Statechart editor we used the Kouretes Statechart Editor ([21]) with two changes. First we changed the Transition Expression attribute to be a multi- line String, done via the .genmodel file. After that we had to remove some of the already existing Audit rules because there were syntax checks in order to have proper code generation for the Monas architecture which is no longer supported by the ASEME IDE as deprecated. To remove already existing Audit rules we had to remove them from the .gmfmap file but also delete by hand the Adapter functions in statechart.diagram.providers .StateChartValidationProvider.java as they are annotated with the generated NOT tag so even if we regenerate the editor these code blocks remain untouched.

3.3 Integration of existing tools/ Functionality extensions

3.3.1 Replacing IAC with Statechart

IAC is pretty much the same metamodel as Statechart with a slight difference : the relation between the Nodes was stated vice versa at the EMF metamodel. As we can see in (figure X), at the IAC metamodel each child Node refers to its father via the FatherOf relationship while in Statechart metamodel each father Node refers to its children via the Children relationship. We removed IAC an its components(.edit, .editor and .diagram plugin projects as IAC had EMF/GMF implementation as all the other models) of our total project setup and since Statechart (along with its components accordingly) was part of the initial setup, the next step was to replace IAC with Statechart to other project that used it. That was seemingly a relatively easy task, since the two models use the same naming conventions, but the replacement had to be applied in several ways and lead to useful conclusions.



(a) IAC meta-model (b) Statechart meta-model

Figure 3.13: IAC and Statechart

At `IAC_EMFgenerator2` plugin(and generally in projects that use xpnand for code generation and have an EMF model as input),which is responsible for the IAC to JADE transformation needs the models .ecore file inside the project folder in order for the xpand templates to work properly, so after replacing the .ecore file we had to replace the imports at the templates. After that, we replaced IAC imports with the equilevant Statechart ones wherever needed to the java code as well. Then as far as the code for transforming liveness formula to Statechart was part of the existing tools, we updated the regular expressions used for the transformation with those available in the code of SRM to Xpdl transformation that had been submitted to extensive debug during the development of SRM2XPDL tool, always over the same syntax criteria. During the debugging process of liveness2Statechart java code we came across a serious bug. The regular expressions available had problem dealing with the tilde(`~`) operator (`|term~|`) and the program crashed whenever this operator used in the formula. We did not solve this problem because we had a lot of other open issues at the moment but knowing that it exists help us set the tilda operator as unsupported at this stage of the ASEME IDE and left this particular bug for future work.

3.3.2 Importing the SRM2BMPN and XPDL tool

SRM2BPMN tool was the first external functionality addition we made to the ASEME IDE. Since a detailed description on how we integrated this tool to our framework is going to be presented to the next chapter we will be limited to the changes we made to the existing code. We used the current project path from ASEME IDE ,given as an input parameter, to the methods used for opening and saving files from the UI menu in order to be synchronized. Also there was a bug during loading SRM models on different operating systems due to different ways of persisting the newline character in the xml (.srm) file. This solved by removing the `"&#xD"` char sequence from the liveness formula attribute of each Role.

3.3.3 IAC to GG

IAC to GGenerator transformation tool was the second external functionality addition we made to the ASEME Model-Driven process. Trying to embed it to the framework there were problems invoking the xpand templates through the workflow.mwe file in runtime, so we did it through java code.

To run the xpand templates through Java we used the xpand language implementation package(org.eclipse.xpand2) , and more specifically XpandExecutionContextImpl, XpandFacade, output.Outlet and output.OutputImpl as well as org.eclipse.xtend.typesystem.emf.EmfMetaModel to define the input model. These imports used in he created statechart2Naoth.Stct2NAoth.java class, in the existing IAC-2-Naoth project, in the public static void generateNaoth method which takes as inputs the desired path for the generated code and the Statechart model to be transformed to generic C++.

Listing 3.5: generateNaoth method

```
public static void genarateNaoth(String srcGenPath, Model statechart){


    OutputImpl out = new OutputImpl();
    Outlet outlet = new Outlet(srcGenPath);
    outlet.setOverwrite(false);
    out.addOutlet(outlet);

    Outlet transitions_outlet = new Outlet(srcGenPath+"/transitions/")
        ;
     transitions_outlet.setOverwrite(false);

    Outlet activities_outlet = new Outlet(srcGenPath+"/activities/");
     activities_outlet.setOverwrite(false);

    Map<String, Outlet> outlets = new HashMap<String,Outlet>();
    outlets.put("default", outlet);
    outlets.put(" activities_outlet ", activities_outlet );
    outlets.put(" transitions_outlet ", transitions_outlet );

    OutputImpl.resolveOutlet(outlets, srcGenPath+"/activities/", "
         activities_outlet ");
    OutputImpl.resolveOutlet(outlets, srcGenPath+"/transitions/", "
        transitions_outlet");

    XpandExecutionContextImpl executionContext = new
        XpandExecutionContextImpl(out, null);
```

```
    // Configure the metamodels
    EmfMetaModel emfMetaModel = new EmfMetaModel();
    emfMetaModel.setMetaModelPackage(StatechartPackage.class.
        getName());
    executionContext.registerMetaModel(emfMetaModel);
    XpandFacade xpandFacade = XpandFacade.create(executionContext);
    Object[] params = null;

    xpandFacade.evaluate("mainTemplate::model", statechart, params);
}
```

After that we modified the dashboard to meet the new requirements (added the GGenerator box, the arrow from IAC and the transform button) and created the equivalent ASEMEAction implementation class for the transformation action on the dashboard that gathers the path and model for the input and calls the above method. This is the standard procedure for adding external functionality to the dashboard with a more detailed description available in next chapter.

3.3.4 Transformations refinements

With the various editings made in EMF metamodels and considering the fact that the existing code had parts that needed debugging, the transformations code had many fixes that their technical detailed description deemed unnecessary. We considered more appropriate to discuss how we extended the existing functionality.

We set as the default behavior of the transform actions to open the diagram editor of the newly created model and for the code generation we added a dialog notifying the user that the transformation was completed successfully. Otherwise if something goes wrong the user is also notified by a dialog with an appropriate message according to the applying audits.

In addition, we added a code block for updating the project explorer when multiple files are generated.

Also, in SUC2SRM and SRM2IAC transformations we automated the import process of information from AIP and EAC respectively. The condition for the auto-include process to begin is the existence of the equivalent file to import from (AIP model with the same name and Statechart file with the

protocol name) in the project folder.

All the changes mentioned above had to be made to ASEMEDashboardview project, and are implemented at asemedashboardview.views.actions package. The actual transformation code is in ASEME_Transformations plugin. Almost all the transformation code has been rewritten as static methods and in accordance with the changes made to the meta-models. We created the class AsemeModelSaveHelper to ASEME_Transformations plug-in in order to automate the persistence of the models. In the class there is a static org.eclipse.emf.ecore.resource.ResourceSet initialized with all the ASEME meta-models registered and two methods for saving models to the given org.eclipse.emf.common.util.URI or filename (java.lang.String) using org.eclipse.emf.ecore.resource.Resource. That is more generally the way we deal with persistence of the models throughout ASEME IDE. Further changes made to the IAC2JADE transformations. We also used Java for invoking the xpand templates, as with GGenerator transformation mentioned above with the addition of the use of org.eclipse.xpand2.output.JavaBeautifier for the format of the generated code provided by xpand. We also fixed a bug in the existing code as the list of the Nodes of a Statechart was not properly created. The problem was that the implementation of the tree structure of the Statechart meta-model the children of each Node is a List of Nodes so we needed a method using a tree2list recursive algorithm to access all nodes of the tree-structure and return all nodes to a static list in order to be visible for other classes where needed. Since the xpand framework does not support recursive code we had to put that method to the java helper classes of IAC_EMF.generator2 project.

3.3.5 SRM to IAC import EAC

At SRM2IAC transformation, SRM.Capabilities are transformed to Statechart.Nodes for each SRM.Role. Capabilities at SRM level might be Protocols at AIP. Since in EAC we have a Statechart for each AIP.Protocol we automated the import of Variables and Transition Expressions, that contain useful info for the code generation, to the appropriate Nodes of the equivalent Role Statechart. We used the algorithm presented in pseudo code below :

Listing 3.6: SRMImportEAC algorithm

```
For each Role.Node
    If (this.Node.TYPE == "OR")
```

```
            If this.Node.isProtocol() {
               Load Protocol
                  Associate Varibles
                 Associate TEs
                 }
              End if
         End if
    End for
```

After we have the generated Statechart models, we needed to post-process each model in order to enrich it with extra info from the SRM model, so an iteration over the tree was necessary and we tried during that single iteration over a Role statechart to also import any protocol info we have. Considering the fact that in order to associate TEs and Variables from a protocol and a role statechart we have to find their common sub-tree and transfer them per Node and thus more iterations over different sub-trees are needed, we tried to be efficient. We used four helper functions

exportCapabilities(SRMModel): List¡String¿

DFsearch(Node, String):Node

AssociateTEs(Model protocol, Model role, String protocolPrefix, String rolePrefix) :void

AssociateVars(Node protocolNode, Node roleNode, Model stct, String protocolPrefix, String rolePrefix): void

(Algorithm using helpers / more technical)

3.3.6 Abstract Role/Protocol support

During the development process and due to the changes made to the SUC meta- model we came up with the idea of treating any Abstract Roles on SUC as System ones, concerning the ASEME Model Driven Process. By supporting Abstract Roles at the SUC model SUC2AIP transformation leads to Abstract Protocols at AIP model. Subsequently, at SRM model Abstract Protocols are imported as independent capabilities.

This admission and its derivatives to AIP and SRM provide additional support of a Protocol between two Abstract Roles and Between Abstract and System Roles and enriches the existing design process.

The automation for Abstract Roles/Protocols supported by the transformation tools in two levels :

- SUC2AIP transformation Abstract Roles are valid for transformation, generating Abstract Protocols at AIP
- import AIP : (SUC2SRM transformation) import Abstract Protocols as standalone Capabilities available for the user to assign to any SRM Role with the protocol liveness formula that can be found in the description field of a capability

3.3.7 Transition to Eclipse Mars Environment / Update Site

For the transition of the application from Eclipse Luna to Mars and later to Mars 2.0 environment we had imported the plugin projects to the new Eclipse instance. Due to incompatibilities we had to regenerate the auto generated code for each model but in order for it to work properly we had to set the Compliance Level property of the .genmodel to 6.0 as shown in (fig). Also there were some problems with the JVM version used by some of the projects and some other minor configuration issues that all could be resolved through the MANIFEST.MF file of each project.

To export ASEME IDE we followed two simple steps. First we created a new feature project. A feature is a collection of plug-ins, in fact it is a xml file describing the plug-in group. Eclipse provides a really helpful ui to edit the feature.xml file similar to manifest.mf files and there we add the the desired plug-ins that constitute the feature, compute their dependencies and so on. If there are any external dependencies to any plug-in must be registered also here to be visible during runtime as shown in (fig).

The second step was to create a new Update Site Project. Both Feature and Update Site Projects are part of the Eclipse wizards in Plug-in Development category accessible via File -> New -> Other. After the creation we add the feature to the update site and add any extra information needed to site.xml file and finally build. The Update site project folder is now ready to be hosted at any server or used as a local repository.

Chapter 4

Interfaces

In this chapter we are going to present the way to add new functionality to the existing ASEME IDE by example, describing the process followed to integrate SRM2XPDL tool to the framework. More specifically, in subsection 4.1 we will describe the procedure of adding a new plugin (or set of plugins for a GMF editor) with the desired new functionality and the appropriate actions that need to be taken to register it properly to the framework, and in subsection 4.2 we will show the steps that need to be followed for registering a new transformation and adding the desired functionality.

4.1 Add new model / editor

When the addition of extra functionality is deemed necessary, the developer must provide the functionality in an Eclipse Plugin. In accordance with those indicated by the Eclipse Plugin Model, any external dependencies must be in the classpath in order to be visible at runtime (Figure 4.1).

Also the package(s) that contain code that needs to be accessible from outside the plugin need to be exported (Figure 4.1) and finally the new plugin needs to be added to the dependency list of our main plugin ASEMEDashboardView (Figure 4.2).

After the standard actions described above the developer needs to alternate the dashboard in order to meet the new requirements. Responsible for modifying the Dashboard visually and adding functionality to it are the classes ASEMEFigure and ASEMEMediator.

At ASEMEFigure java class we define the layout and parts of the Dash-

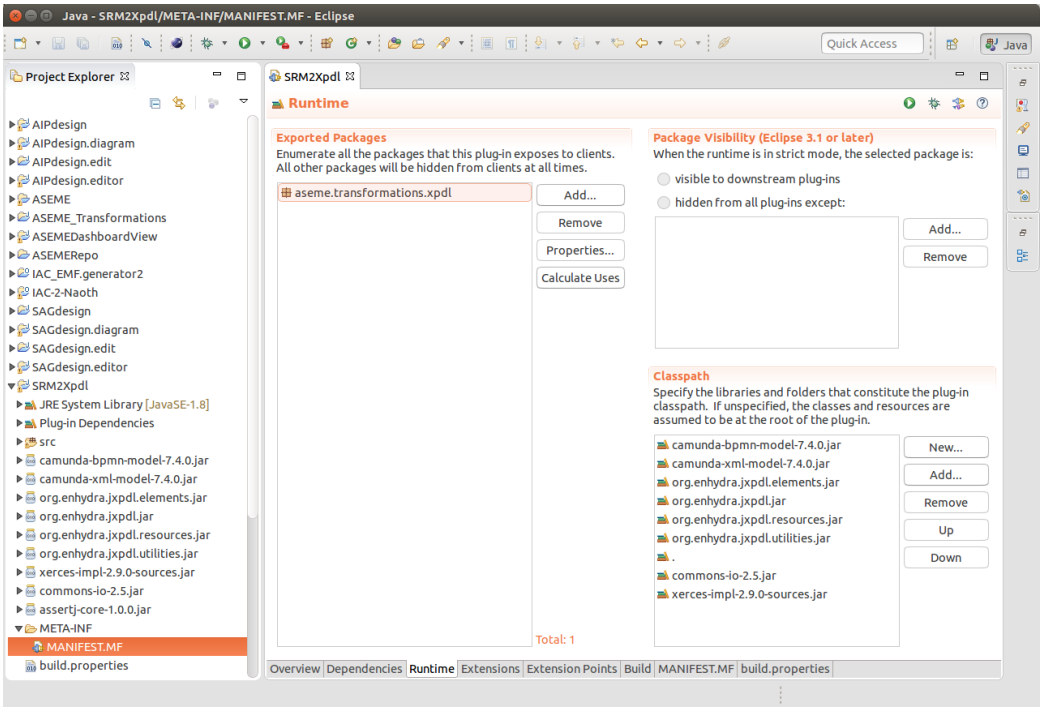


Figure 4.1: SRM2XPDL/MANIFEST.MF runtime tab

board in terms of a figure. As the Dashboard is implemented extending draw2d (`https://www.eclipse.org/gef/draw2d/`) RectangleFigure, it can be customized using org.eclipse.draw2d.geometry package in order to arrange the figures that constitute the dashboard at will.

There are three main figures that are used repeatedly for the visual construction of the dashboard and each of them is imple- mented as a java class in asemedashboardview.views package : ModelFigure, FlowFigure and FlowActionFigure.

Each model of ASEME is represented by a box in the dashboard and it is implemented as a ModelFigure. The ModelFigure class extends draw2d RectangleFigure(same as the dashboard) and implements ActionContainer interface that gives the ability to have the actions (Create, Select, Edit)

For the edges connecting the ModelFigures we have the FlowFigure class that extends Polyline and implements Connection, both from draw2d thus customization is possible as stated by the rendering toolkit.

Our last main figure is the FlowActionFigure, which visually is the box

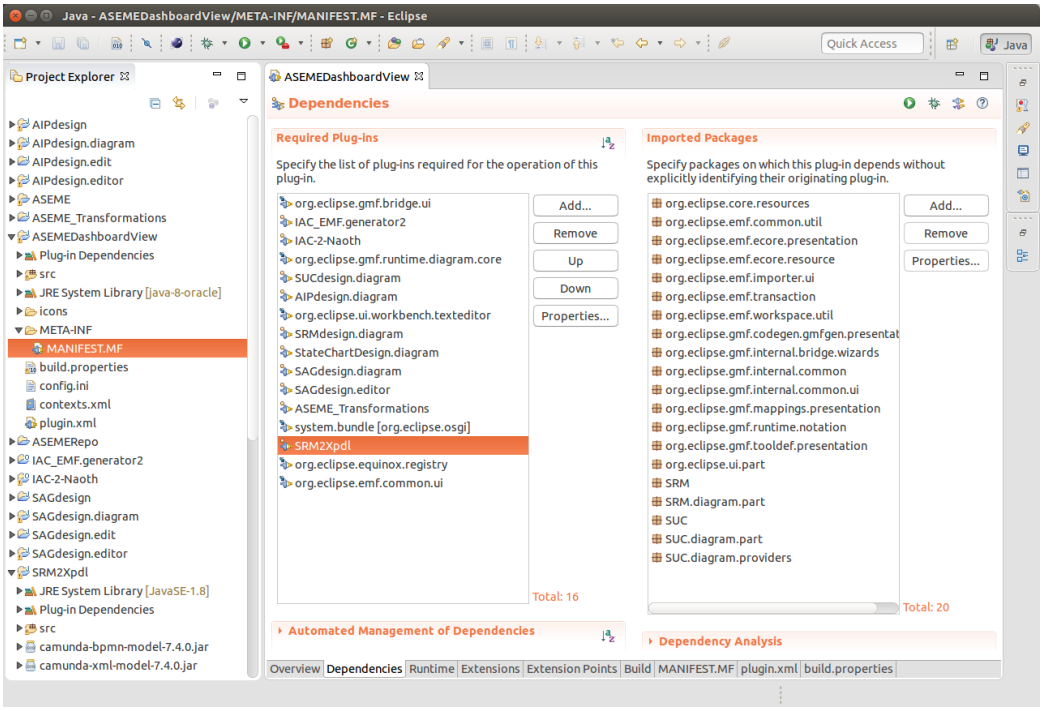


Figure 4.2: ASEMEDashboard/MANIFEST.MF dependencies tab

with the transform label on the FlowFigures. This class also extends draw2d RectangleFigure and implements ActionContainer interface so it can support the actions for the transformations.

In this case we follow the steps below : add the new box entitled "Process Model (BPMN 2.0 or XPDL)" (ModelFigure) define the actions on the bottom side of the box(select and edit) add the arrow from SRMBox to the new Box and (FlowFigure) add the transform label (define the action) (FlowActionFigure)

4.2 Add new transformation

When all the steps described above have been done the process to add a new transformation is relatively easy. First we make a new .java file (a class) at ASEMEDashboardView plug-in in views.actions package that implements the ASEMEAction interface and in its run() method we add the code that will run when the transform button is pressed in the dashboard (4.3).

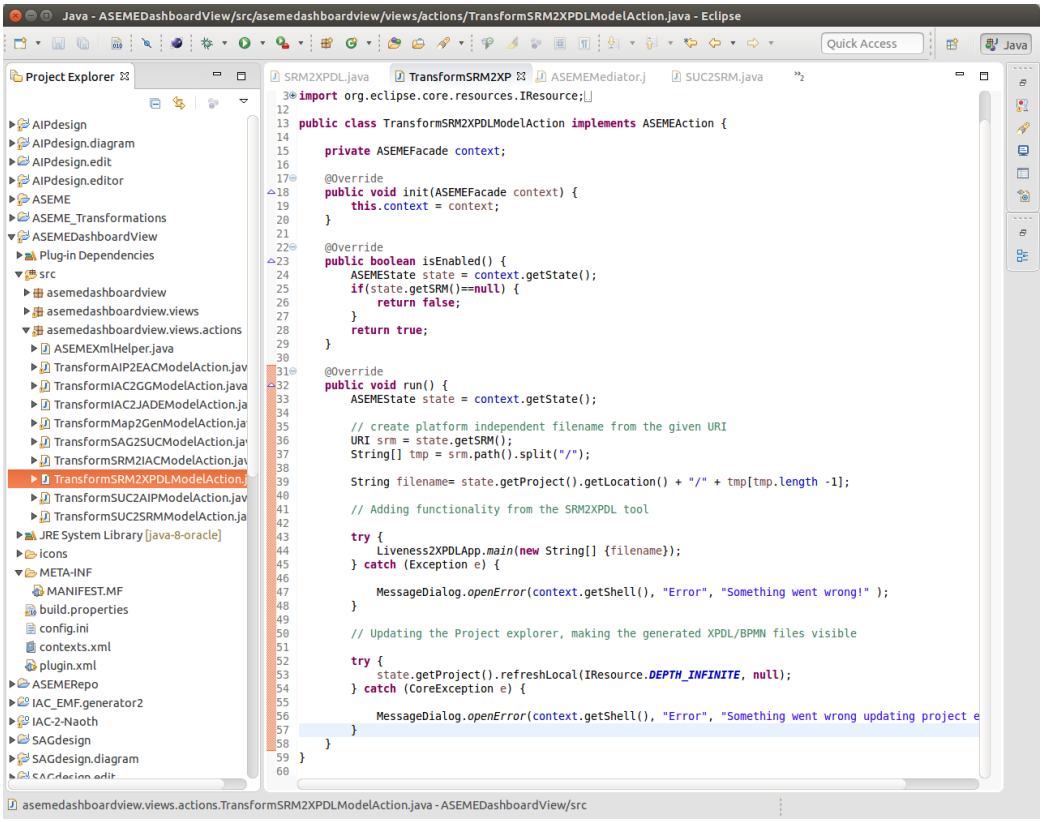


Figure 4.3: ASEMEAction interface implementation for SRM2XPDL transformation

Then, mostly for modularity reasons, we follow our design and make a .java class at ASEMETransformations plug-in where we put a static method that implements the desired functionality (in this case it just launches the external app) and we call it from the ASEMEAction class *run()* method (Figure 4.4).

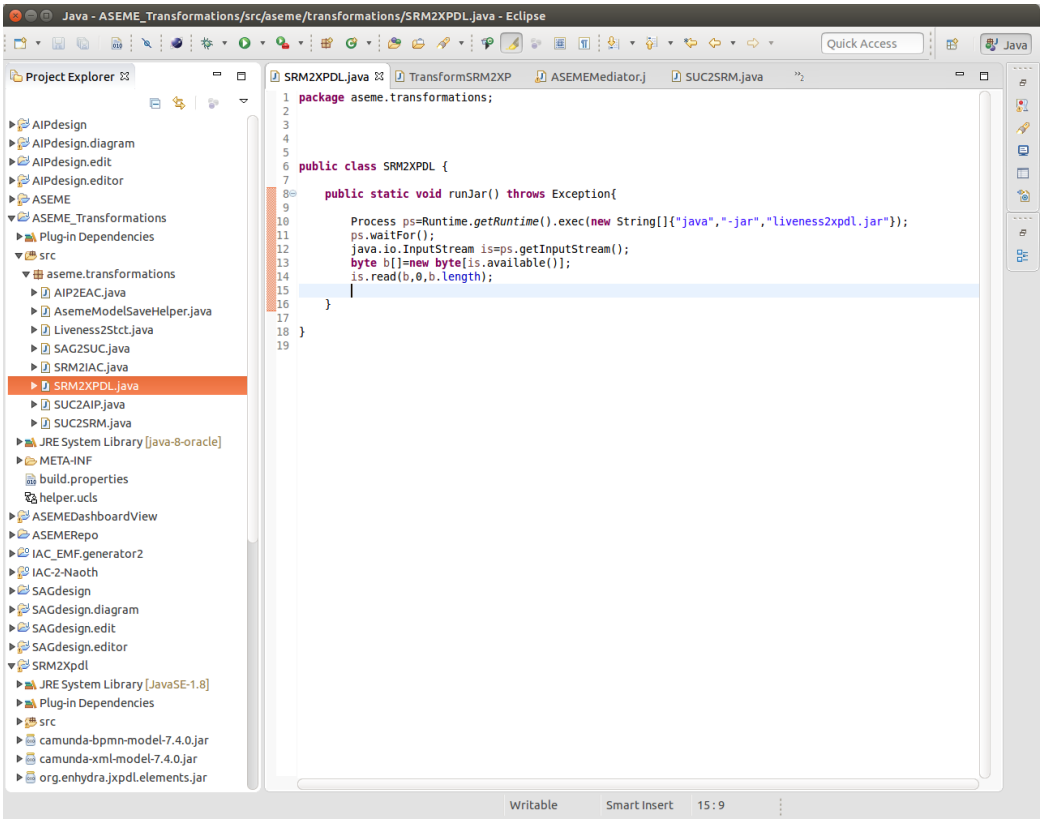


Figure 4.4: SRM2XPDL

Chapter 5

An example of the ASEME modeling process

In this chapter we are going to present an example on how to use the ASEME IDE. We will go through each step of the Model-Driven process with detailed comments on the usage of the current implementation of the IDE and also discussing on what the user should note about the behavior of the tool.

5.1 A negotiation Agent

For this example we are going to design and implement a simple negotiation Agent that uses argumentation based decision making policies that negotiates with a protocol defined by another argumentation based theory.

In order to start working with the ASEME IDE a user has to download the Eclipse Modeling Tools Mars 2 `http://www.eclipse.org/downloads/packages/eclipse-modeling-tools/mars2` . After installing, two additional modeling components are required : Xpand and Graphical Modeling Framework Tooling, that are available through the Install Modeling Components option of the toolbar.(5.1)

Install the ASEME new software by adding the ASEMERepo (http://aseme.tuc.gr/aseme/ASEMERepo/) update site (from the menu Help -> Install new software and by clicking the Add button) Then start a new general project and show the ASEME Dashboard view (from the menu Window -> Show view -> Other and from the list select the ASEME -> ASEME Dashboard)

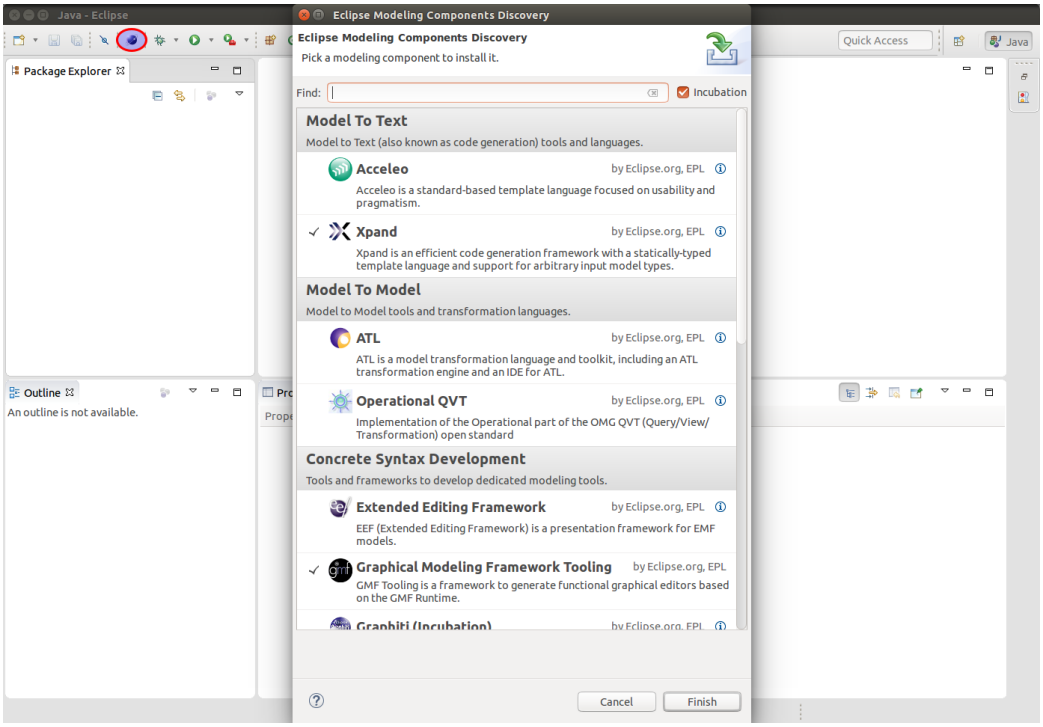


Figure 5.1: Install Modelling Compoents for ASEME IDE

5.2 Requirements Analysis

An agent during his life cycle may assume many roles. At this stage we have to analyze our system in a societal level, thus to define the Actors being present, and their Goals. After defining the possible agent roles and their goals we analyze the requirements per goal and decide on possible interactions.

In this particular example we need an agent that given a negotiation protocol can use its own knowledge to negotiate with another agent and another one providing the protocol. So we have two Actors : a Negotiator and a ProtocolKeeper.

Now we are going to represent this in our SAG model. After clicking the create button in the SAG box of the ASEME Dashboard an empty SAG model is created and the GMF editor is opened. We can add items from the palette to the editor(as in any graphical editor) and we define our two roles with their goals as shown in (5.2).

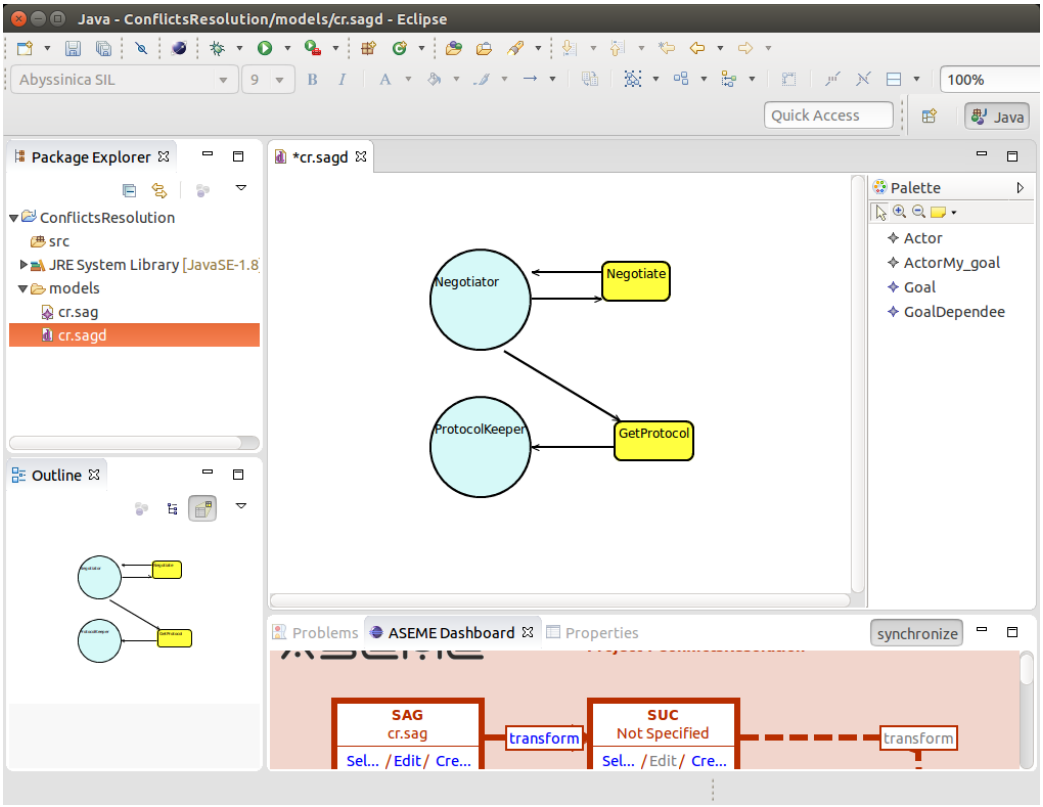


Figure 5.2: example SAG model

5.3 Analysis

Preceding with the analysis phase we continue with the SUC model.

For each model transformation we just click on the transform label on the arrow between the two model figures(boxes) on the dashboard and then the derived model is created and the an instance of the corresponding editor is opened with the newly created model ready to be edited.

In our example the derived SUC model looks like this(5.3). At the SUC level we can add more information about the use cases and add any additional roles that are needed to design our system in more detail.

At this stage of the analysis we come up to the first optional part of the ASEME Model Driven process which is the use of the AIP model for the design process. Usually the use of AIP is recommended because AIP is responsible for the more detailed description of the protocols.

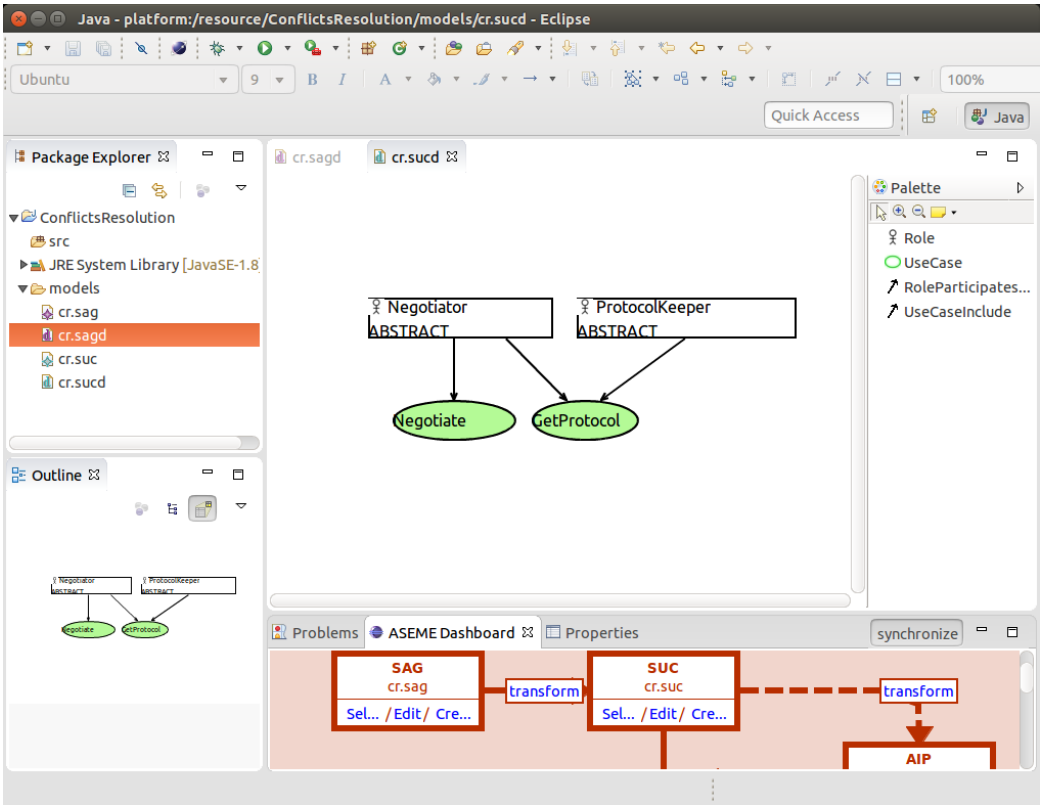


Figure 5.3: derived SUC model

SUC to SRM transformation requires only the SUC model but if AIP also exists (and is selected in the dashboard) will also be used by the transformation algorithm. For this example we are going to present two different ways on how the information of SUC and AIP models are combined to produce the SRM and how it affects the auto-generated SRM model.

At this point we considered appropriate to present some notes on the transformation algorithm :

- A use case that has two or more roles as participating roles will be transformed to a protocol to AIP
- Each use case that includes other use cases will be transformed to a capability to the SRM model
- the included use cases constitute the activities of the capability, thus

their apposition with the appropriate operators consist the behavioral description of the capability, its liveness formula.

- the liveness formula of a capability is part of the SRM Role liveness formula and is also the Liveness property of a Participant in the AIP model
- the generated formula in both cases described above has `?OP?` as sample text where the operators are expected
- only System roles will at SUC will be transformed to SRM Roles
- any participants liveness formula at AIP will be imported at SRM either in the formula of the appropriate Role (if its a System Role) or as the description field of a standalone capability (in case of Abstract Roles/Protocols)

The first option is to enrich the SUC model with information and describe in depth the desired behavior by extending the use cases as shown in figure(5.4). Then we have the generated AIP model in 5.5 and after the placement of the desired operators the SRM model of 5.6 is generated.

The second option we will present here is to only add two Abstract roles at the generated SUC model as participants to a use case 5.7. This will provide a similar AIP as above with the difference that the liveness property of each participant will be empty because the use-case analysis in SUC is absent. So by writing the liveness formula for each participant to define the desired behavior(in both cases of our example we use the same protocols so the same formula but in second case Negotiation Protocol is abstract 5.8) we get generated SRM shown in 5.9 which has almost the same information as in the previous case in a slightly different, more flexible format.

At the SRM the modeler has the possibility to enrich the model in various ways, as capabilities and activities can be reused in more than one roles and many activities can have the same functionality (eg web service invocation, algorithm etc). The reader should note that any changes on the graphical editor will **not** be automatically reflected to the liveness formula of the role and must be added there by hand(all editings done through the properties view 5.9). This feature is not supported yet by the ASEME IDE and the transformations that have the SRM model as input (SRM to BPMN and SRM to IAC) rely solely on the liveness formula of each role.

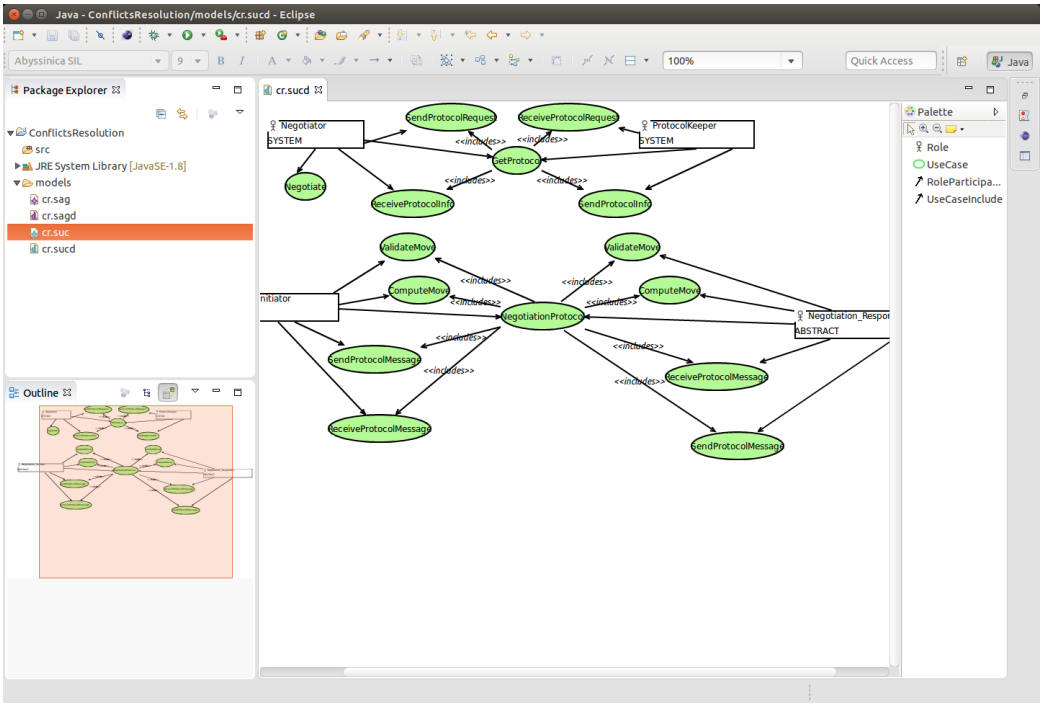


Figure 5.4: SUC model refined

By hitting the transform button and launching the Liveness to BPMN/XPDL application. The working directory is provided as root for the application for the input actions and the produced files are also saved to the project folder. For the interested reader, further details on the transformation algorithm are presented to ([7]) or in the ASEME IDE repository (repo) in the model folder.

5.4 Design

From the liveness formula of each SRM role a Statechart model is generated. The auto-generated models(.stct) are located in the project folder along with one .kse file each(5.10).

Changes here will be affecting the code since is the last model used in ASEME MDP before code generation, thus here the modeler can design in detail the implementation of the behaviour of each role that an agent would assume in our system.

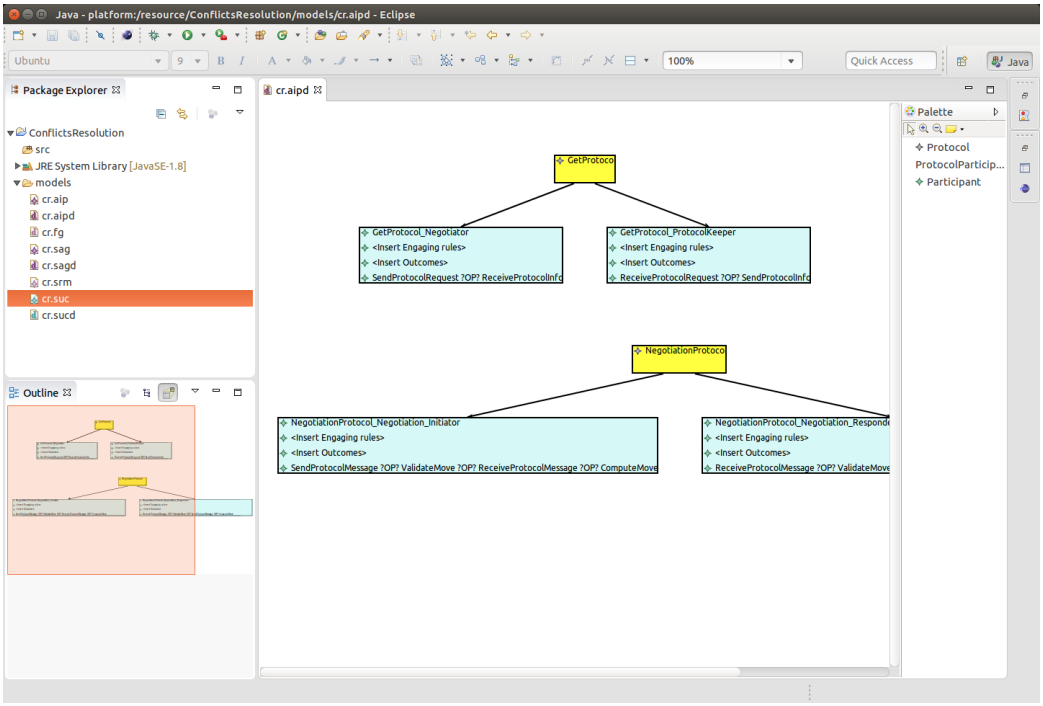


Figure 5.5: derived AIP model

Statechart model generation is also provided from the liveness formula of each protocol at AIP model(AIP2EAC).

If protocol statecharts have been generated before, any information available in the protocol statechart will be imported at each role statechart that participates in each protocol.

For our example we present our final version Negotiator role statechart in 5.11 and the state variables in 5.12 (the assignment and editing of variables to a node is done via the properties view as shown in figure). Note that the Transition Expressions(TEs) and Variables (also the association between Variables and States) shown in the figures are automatically imported from the Protocol statecharts (EAC).

5.5 Implementation-code generation

The last two steps are for code generation, so by hitting the transform button for IAC2JADE or IACtoGenericC++ the generated code can be found at src-

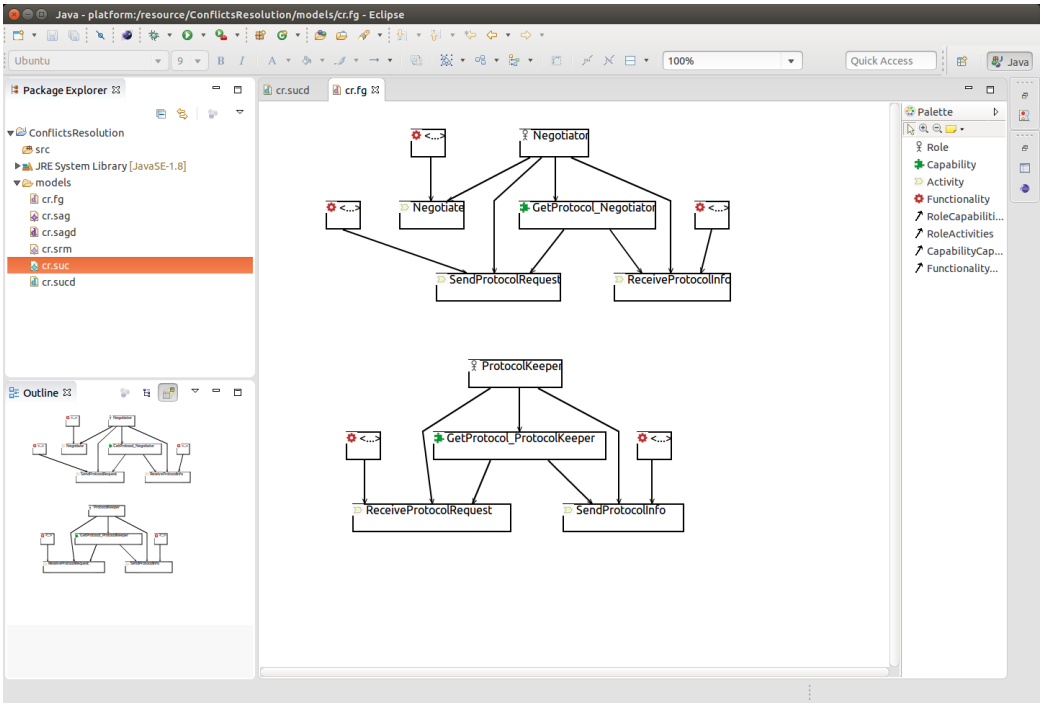


Figure 5.6: derived SRM model

gen/JADE and src-gen/NAOTH respectively as shown in figure(). From this point on the user may customize directly the generated code.

5.6 Important Notes

ASEME IDE users must note that the dashboard is not automatically updated according to the project explorer(eg if a model is selected and delete it from the project explorer it will **not** be removed from the dashboard). This bug exists also in the GMF Dashboard implementation and during the development process it was not considered a must-have feature(mostly for time management reasons) but due to its importance for the user experience we have to point it out for the users and rank it high in the future work priority list.

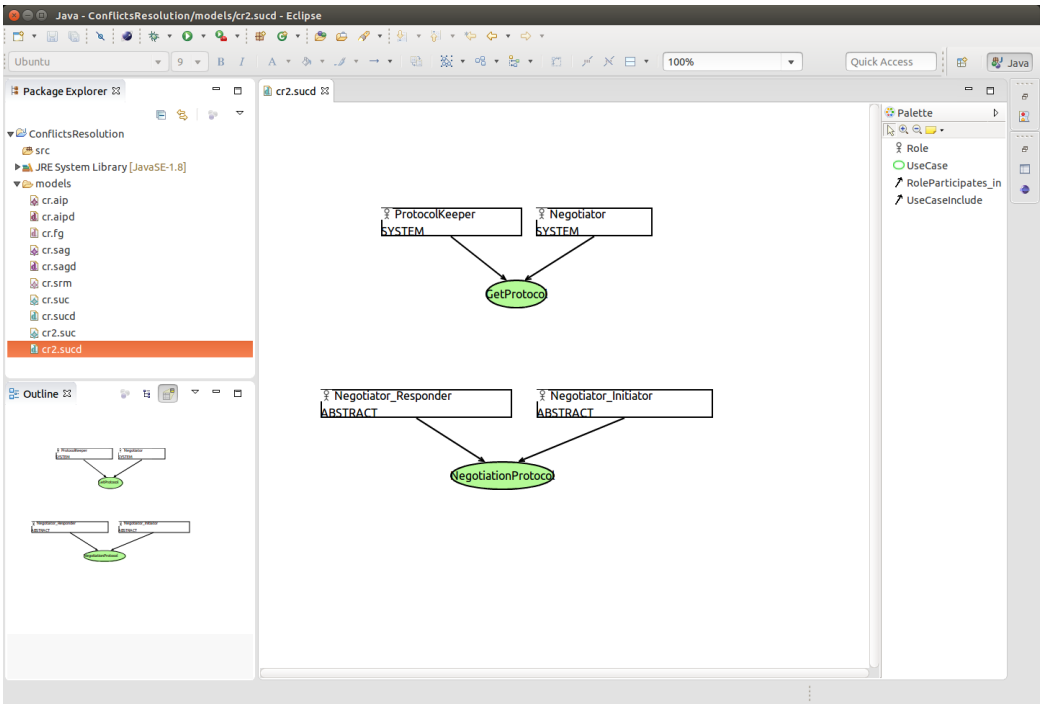


Figure 5.7: SUC with Abstract Roles

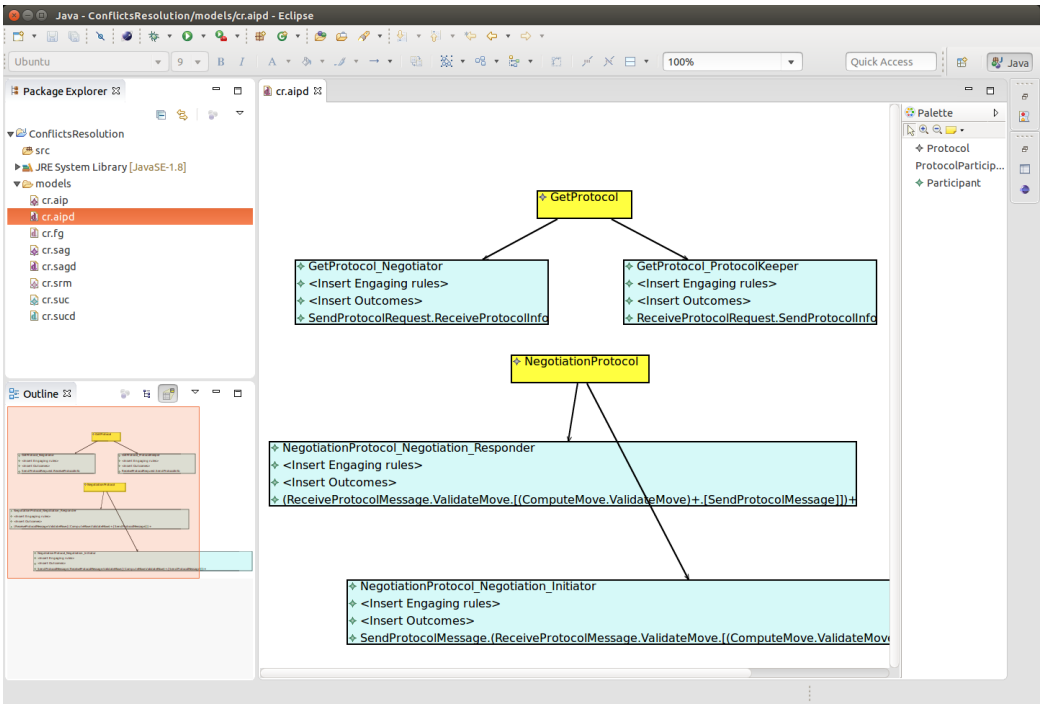


Figure 5.8: refined AIP model

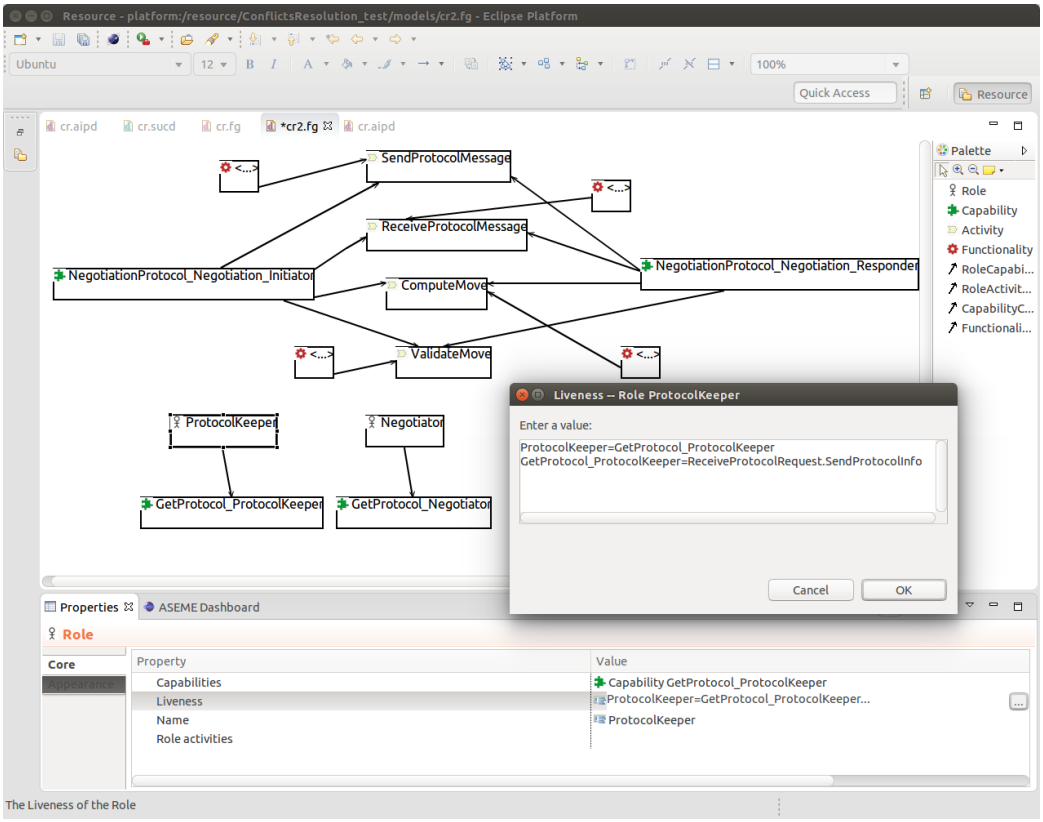


Figure 5.9: refined AIP model

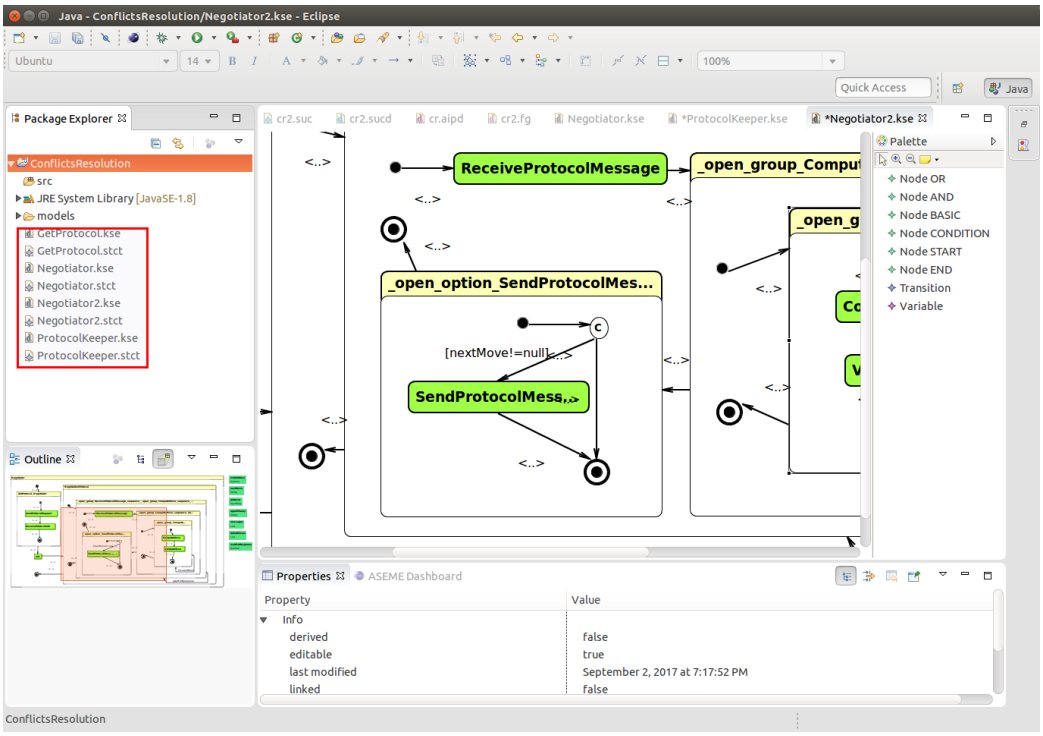


Figure 5.10: Generated Statechart models

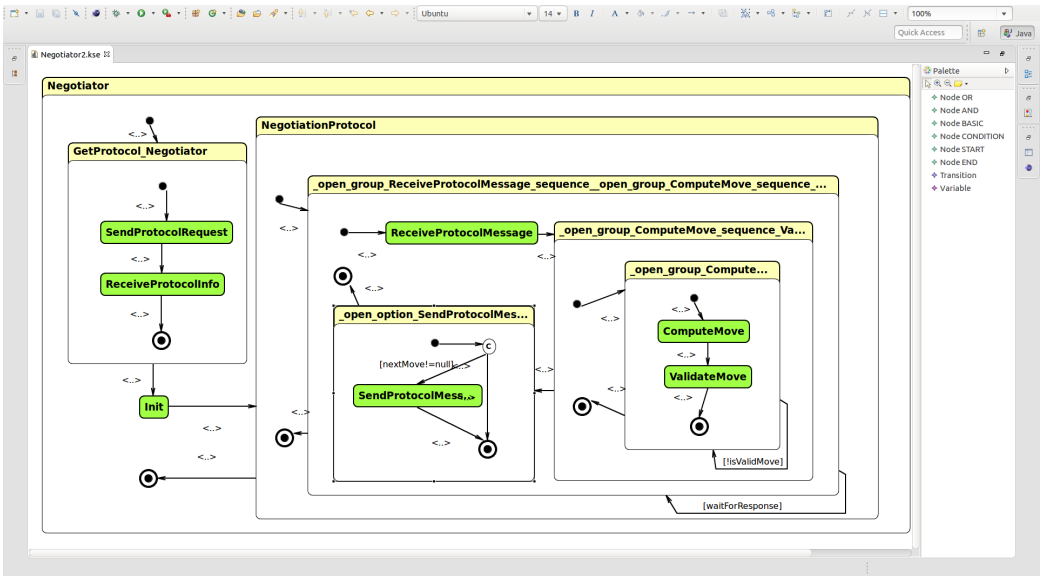


Figure 5.11: Statechart model for Negotiator

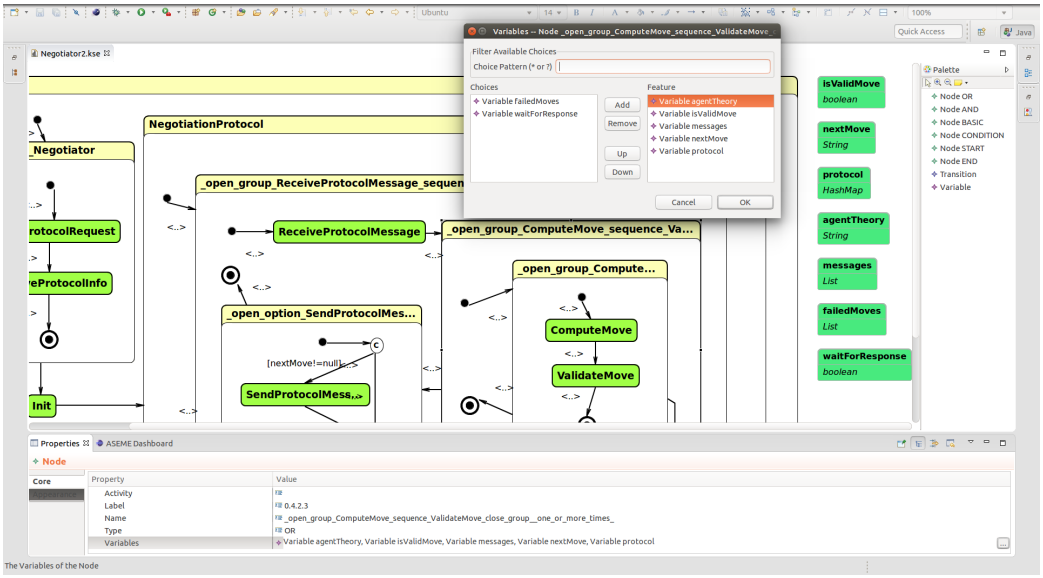


Figure 5.12: Negotiator statechart variables

Chapter 6

Conclusion

6.1 Discussion

In this thesis we presented ASEME IDE, an advanced case-tool that provides a user the possibility of designing a multi-agent system (MAS) using the ASEME methodology from scratch and going through a model driven process to either export the designed system to business models (BPMN/XPDL) and/or code generation (Java/C++).

More specifically, we brought together various previous works on ASEME (the AMOLA metamodels and transformation tools[14], Kouretes Statechart Editor (KSE) [20] [21], an extension of KSE for Executing Statechart-Based Robotic Behavior Models [9] [10] and an application to transform SRM to BPMN/XPDL models [6] [7]), integrating into one application that could support the entire ASEME model-driven process and created a dashboard to guide the user through. During the development process, we had to modify the existing code and completely rewrite parts of it mostly due to the refinements we made to some of the AMOLA meta-models implementation. We also made functionality additions and extensions that finally led us to be able to support an enhanced version of the ASEME process. ASEME IDE is implemented as a plug in for the popular Eclipse platform and we also provide an update site to install from.

Towards this thesis completion, there were some parts of the process that have not been discussed yet. For example the addition of the SRM2BPMN Swing application to ASEME IDE was a time-consuming task. The code integration part of the process was relatively easy but the configuration part

was more complicated. The visibility of the external dependencies must be ensured in every level with the most important of them being the top level, when we group our plugin projects to a feature. If the desired extended dependencies are not declared for exportation, when the ASEME IDE is installed at an Eclipse instance and a user triggers (via the transform button in our case) the application to start, ASEME IDE crashes and the logging message is misleading in order to find the cause of this problem. So in future functionality addition/extensions a developer must ensure that the steps detailed in Chapter 4 and Section 3.3.7 are followed to avoid unpleasant surprises and save time.

At some point we also tried to provide an alternative view for editing the models (mostly SRM and AIP) using popular EMF Forms Framework (`http://www.eclipse.org/ecp/emfforms/`). After multiple tries on several versions of EMF Forms we did not succeed in having the desirable outcome mainly because of the inability of the framework to support the simultaneous editing of children entities included at a parent entity (eg Protocol Participants at AIP, Role Capabilities at SRM). It may be possible to do succeed at this task since we are not aware of the current status of the framework but we have to mention that developer support is charged with a fee rather large for a thesis!

Concerning the tools that used through this thesis we have to note that based on our experience of developing and customizing the editors, GMF generated editors have many out of the box features that are helpful but when it comes to add custom code, although at many cases the addition of few lines of code was required, the time needed to complete such a task was disproportionate with the amount of code needed. Thus, we have to agree with [12] that GMF does shorten development time, but to a smaller degree than we initially expected, especially for someone with no prior experience with DSL editor frameworks.

Concluding, we consider that (mostly thanks to the dashboard approach) we succeeded towards the outcome of this thesis, ASEME IDE, being is a complete software tool that is more user friendly than the underlying concepts, but as always there is a lot of room for improvements!

6.2 Future work

During the development of ASEME IDE among the features added and the bugs resolved there were a lot of features that either we did not have time to implement or considered nice to have and not must have features. Also due to the process made other improvements and features have emerged.

An improvement that will make ASEME IDE much more user friendly is the use of GMF audit rules in all model editors (for now some audit rules are used in KSE). More generally anything that would guide the user in a more specific manner during the design process will be a nice add-on. For example we tried to implement live validation of the SRM model GMF editor. More specifically we tried to have the editor updated on every liveness formula change (re-assign capabilities and activities or create them if they do not exist and vise versa) but din not have the time to complete it and have it in the current release. Our advise to anyone that would like to be involved with that feature is to create a grammar (using xtext) for the formula validation.

Another nice-to-have feature would be the auto generation of simple form protocols at the SUC editor. This idea came from the send-receive pattern observed in most of the cases of defining a protocol between two actors in a MAS. An interesting idea that could help for the implementation of this feature can be found at [19].

A feature that will update the whole ASEME IDE is to add code generation for SRM functionality or even for SRM capabilities. Especially if the generated code is stored by the IDE it takes the ASEME MDP to a whole new level as we can have a pool of reusable code to customize the agents behaviour. For example the generated code from functionality code can be the invocation of a web service and extending it to capability generated code can have also an implemented algorithm over the response of the web service invoked.

6.3 Lessons learned

While working towards the completion of this thesis I learned that a lot of reading is required before you start coding, especially when you have to work with advanced software architectures, so keeping notes and in general good organization is really helpful. Also due to the fact that you will have to build upon already existing work of other people reading the manual is essential

while the next step is to contact the author. I also realized that it is a major responsibility for any author (including me in that thesis) to try and write a manual or any other text on what he did that is relatively easy to understand in order to cause the least trouble possible to anyone that would need his work afterwards. Last but not least I realized that time management is essential, but is also a really common problem at any development process. It always take more time than your original estimation.

Bibliography

[1] Jos Warmer Anneke Kleppe and Wim Bast. *Practical Eclipse Rich Client Platform Projects*. Addison-Wesley Longman Publishing, 2003.

[2] Henderson-Sellers B. and Giorgini P. "Agent-Oriented Methodologies". In: (2005).

[3] Richard C. Gronback. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*. Pearson Education, 2009.

[4] B. Henderson-Sellers. *Agent-Oriented Methodologies*. IGI Global research collection. Idea Group Pub., 2005. ISBN: 9781591405870. URL: https://books.google.gr/books?id=ylGAvAA3dPUC.

[5] Nicholas R Jennings and Michael Wooldridge. "Agent-Oriented Software Engineering." In: *MAAMAW*. Citeseer. 1999, pp. 1–7.

[6] Nektarios Mitakidis. "Transforming ASEME Roles Models to Process Models". Technical University of Crete, 2014.

[7] Mitakidis Nektarios, Delias Pavlos, and Spanoudakis Nikolaos. "Validating Requirements Using Gaia Roles Models". In: *Revised, Selected, and Invited Papers of the Third International Workshop on Engineering Multi-Agent Systems - Volume 9318*. New York, NY, USA: Springer-Verlag New York, Inc., 2015, pp. 171–190. ISBN: 978-3-319-26183-6. DOI: 10.1007/978-3-319-26184-3_10. URL: http://dx.doi.org/10.1007/978-3-319-26184-3_10.

[8] Object Management Group (OMG). *Meta-Object Facility (MOF) Specification, Version 2.0*. OMG Document Number formal/2006-01-01 (http://www.omg.org/spec/MOF/2.0). 2006.

[9] Georgios Papadimitriou. "Extending Kouretes Statechart Editor for Executing Statechart-Based Robotic Behavior Models". Technical University of Crete, 2014.

[10] Georgios L. Papadimitriou, Nikolaos I. Spanoudakis, and Michail G. Lagoudakis. "Extending the Kouretes Statechart Editor for Generic Agent Behavior Development". In: *Artificial Intelligence Applications and Innovations: 10th IFIP WG 12.5 International Conference, AIAI 2014, Rhodes, Greece, September 19-21, 2014. Proceedings*. Ed. by Lazaros Iliadis, Ilias Maglogiannis, and Harris Papadopoulos. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 182–192. ISBN: 978-3-662-44654-6. DOI: 10.1007/978-3-662-44654-6_18. URL: http://dx.doi.org/10.1007/978-3-662-44654-6_18.

[11] Douglas C Schmidt. "Model-driven engineering". In: *COMPUTER-IEEE COMPUTER SOCIETY-* 39.2 (2006), p. 25.

[12] Fredrik Seehusen and Ketil Stølen. "An Evaluation of the Graphical Modeling Framework (GMF) Based on the Development of the CORAS Tool". In: *Theory and Practice of Model Transformations: 4th International Conference, ICMT 2011, Zurich, Switzerland, June 27-28, 2011. Proceedings*. Ed. by Jordi Cabot and Eelco Visser. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 152–166. ISBN: 978-3-642-21732-6. DOI: 10.1007/978-3-642-21732-6_11. URL: https://doi.org/10.1007/978-3-642-21732-6_11.

[13] Vladimir Silva. *Practical Eclipse Rich Client Platform Projects*. Apress, 2009.

[14] Nikolaos Spanoudakis. "The Agent Systems Engineering Methodology (ASEME)". PhD thesis. Paris Descartes University, 2009.

[15] Nikolaos Spanoudakis and Pavlos Moraitis. "Agent Systems Engineering Methodology: The Development Process". In: *VI Agent-Oriented Software Engineering Technical Forum, Bath, UK, December 17*. 2008. URL: http://www.pa.icar.cnr.it/cossentino/AOSETF08/docs/AOSE_TF_08_SpanoudakisMoraitis.pdf.

[16] Nikolaos Spanoudakis and Pavlos Moraitis. "The Agent Modeling Language (AMOLA)". In: *Artificial Intelligence: Methodology, Systems, and Applications: 13th International Conference, AIMSA 2008, Varna, Bulgaria, September 4-6, 2008. Proceedings*. Ed. by Danail Dochev, Marco Pistore, and Paolo Traverso. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 32–44. ISBN: 978-3-540-85776-1. DOI: 10.1007/978-3-540-85776-1_4. URL: https://doi.org/10.1007/978-3-540-85776-1_4.

[17] Nikolaos Spanoudakis and Pavlos Moraitis. "Using ASEME Methodology for Model-Driven Agent Systems Development". In: *Agent-Oriented Software Engineering XI: 11th International Workshop, AOSE 2010, Toronto, Canada, May 10-11, 2010, Revised Selected Papers*. Ed. by Danny Weyns and Marie-Pierre Gleizes. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 106–127. ISBN: 978-3-642-22636-6. DOI: 10.1007/978-3-642-22636-6_7. URL: http://dx.doi.org/10.1007/978-3-642-22636-6_7.

[18] Arnon Sturm and Onn Shehory. "The Landscape of Agent-Oriented Methodologies". In: *Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks*. Ed. by Onn Shehory and Arnon Sturm. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 137–154. ISBN: 978-3-642-54432-3. DOI: 10.1007/978-3-642-54432-3_7. URL: https://doi.org/10.1007/978-3-642-54432-3_7.

[19] Gabriele Taentzer et al. "Generating Domain-Specific Model Editors with Complex Editing Commands". In: *Applications of Graph Transformations with Industrial Relevance: Third International Symposium, AGTIVE 2007, Kassel, Germany, October 10-12, 2007, Revised Selected and Invited Papers*. Ed. by Andy Schürr, Manfred Nagl, and Albert Zündorf. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 98–103. ISBN: 978-3-540-89020-1. DOI: 10.1007/978-3-540-89020-1_8. URL: https://doi.org/10.1007/978-3-540-89020-1_8.

[20] Angeliki Topalidou-Kyniazopoulou. "A CASE (Computer-Aided Software Engineering) tool for robot-team behavior-control development". Technical University of Crete, 2012.

[21] Angeliki Topalidou-Kyniazopoulou, Nikolaos I. Spanoudakis, and Michail G. Lagoudakis. "A CASE Tool for Robot Behavior Development". In: *RoboCup 2012: Robot Soccer World Cup XVI*. Ed. by Xiaoping Chen et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 225–236. ISBN: 978-3-642-39250-4. DOI: 10.1007/978-3-642-39250-4_21. URL: http://dx.doi.org/10.1007/978-3-642-39250-4_21.

[22] Michael Wooldridge. *An Introduction to MultiAgent Systems*. 2nd. Wiley Publishing, 2009. ISBN: 0470519460, 9780470519462.

[23] Michael Wooldridge and Nicholas R. Jennings. “Intelligent Agents: Theory and Practice”. In: *Knowledge Engineering Review* 10 (1995), pp. 115–152.