Master of Science Thesis in Global Software Engineering

Horizontal transformations for models reuse and tool chaining

Stefano Cucchiella

Department of Computer Science
Università degli Studi de L’Aquila, Italy
Mälardalen University, Västerås, Sweden
E-mail: stefano.cucchiella@gmail.com

December 2010
... To my city and its people, struck by an earthquake on 6th April, 2009
Acknowledgements

During my academic studies, many people helped me to achieve my own goals. I really appreciated every single act that all those people did both directly and indirectly.

First, I would thank my entire family who help to make my dreams come true through their love and moral support.

Then, I would thank my supervisor Antonio Cicchetti who strongly supported me during my last year at Mälardalen University through his indispensable practical and technical advices, suggestions on the thesis process and his professionalism in education. I would extend my gratitude also to my supervisor at Ericsson, Pär Asplund, who made me feel how big companies actually work in a shared, worldwide environment.

In addition, I would thank Henry Muccini and Alfonso Pierantonio from the University of L’Aquila, who gave me the possibility to achieve my goals by taking part in the Global Software Engineering European Master (GSEEM) program and their strong education on Model-Driven Engineering field.

Finally, I would thank all my friends spread all over the world for all study days and funny nights, also because they taught me how to live in a wide, multi-cultural environment where borders are over crossed.

What I am today is because of you!

Thank you all!
Abstract

Nowadays industrial software development faces an increasing system complexity together with the necessity to significantly decrease costs and time of the development process and to release at the same time high-quality products. As a consequence, they typically adopt a constellation of proprietary tools each of which dealing with particular stages of the overall development process, namely design, testing, and deployment to mention a few. Model-Driven Engineering techniques are gaining a growing interest as an efficient approach to tackle the current software intricacy. However, the use of a multitude of proprietary tools requires the redundant specification of characteristics of the system and hampers their chaining.

This thesis work is founded on the necessity to provide a horizontal transformation between concepts defined into Executable and Translatable UML (xtUML) models, and semantically-equivalent concepts defined into Rational Software Architect (RSA), through a pivot profile developed onto the Papyrus platform. The proposed profile is defined according to Foundational Subset for Executable UML Models (fUML), an Object Management Group’s (OMG) standard that provides a virtual machine for model execution aims. Since the fUML restriction forces models to be defined slightly different to the UML standard, all necessary stratagems have been analyzed to provide a compliant pivot profile. Finally, a well-known case study is taken into consideration in order to evaluate the modeling choices adopted during the profile’s specification.

Keywords: Model-Driven Engineering (MDE), Tool Chaining, Profile, fUML, Action Language, Papyrus, xtUML.
# Contents

Acknowledgements .................................................................................................................. 3

Abstract .................................................................................................................................. 4

Contents ................................................................................................................................... 5

List of Figures ............................................................................................................................ 7

1 Introduction .......................................................................................................................... 8

1.1 Environment ...................................................................................................................... 9
1.2 Objective .......................................................................................................................... 9
1.3 Literature ........................................................................................................................ 9
1.4 Convention ....................................................................................................................... 9
1.5 Audience .......................................................................................................................... 9
1.7 Thesis Outline .................................................................................................................. 10

2 Background ......................................................................................................................... 11

2.1 Model Driven Engineering .............................................................................................. 11
2.2 UML extension mechanism: go heavy or light? ............................................................... 13
2.3 Object Constraint Language (OCL) ............................................................................. 14
2.4 Executable UML: xtUML .............................................................................................. 15
2.4.1 The Class Model ........................................................................................................ 17
2.4.2 The State Model ......................................................................................................... 19
2.5 Foundational UML ........................................................................................................ 19
2.6 UML Action Language: ALF ...................................................................................... 22

3 Tools differences .................................................................................................................. 24

3.1 BridgePoint .................................................................................................................... 24
3.2 Rational Software Architect ......................................................................................... 25
3.3 Papyrus .......................................................................................................................... 26
3.4 An Evaluation ................................................................................................................ 27

4 Tool Chaining: the pivot profile .......................................................................................... 29

4.1 Primitive Types ............................................................................................................. 29
4.2 Structural concepts ........................................................................................................ 30
4.2.1 The bpSubsystem stereotype ..................................................................................... 31
4.2.2 The bpDatatypes stereotype ..................................................................................... 31
4.2.3 The bpClass stereotype ............................................................................................. 32
4.2.4 The bpExternalEntity stereotype ............................................................................. 33
4.2.5 The bpAttribute stereotype ...................................................................................... 33
4.2.6 The bpRelationship stereotype ............................................................................... 34
4.2.7 The bpGeneralization stereotype ........................................................................... 37
List of Figures

Figure 2.1: The Model-Driven Architect (MDA) approach 11
Figure 2.2: The MDA approach during software development life cycle 12
Figure 2.3: The Meta-modelling levels 13
Figure 2.4: The xtUML development process 15
Figure 2.5: The xtUML process 16
Figure 2.6: Example: student advised by a teacher from his same department 18
Figure 2.7: Translation to and from fUML 20
Figure 2.8: UML Compliance Levels 21
Figure 2.9: The ALF Syntax Elements 23
Figure 3.1: The BridgePoint development process 24
Figure 3.2: IBM Rational Software Architect framework 25
Figure 3.3: Papyrus Framework 27
Figure 4.1: Profile's Primitive Types 30
Figure 4.2: The abstract bpPackage stereotype 31
Figure 4.3: The bpClass stereotype 33
Figure 4.4: The abstract bpPropertyString stereotype and its generalization 34
Figure 4.5: The bpRelationship stereotype 36
Figure 4.6: How to replace the AssociationClass concept in the profiled model 36
Figure 4.7: The bpGeneralization stereotype 38
Figure 4.8: The bpParameter stereotype 39
Figure 5.1: The profiled workbench 41
Figure 5.2: The Microwave Oven packages 43
Figure 5.3: The The Microwave Oven datatypes 43
Figure 5.4: The inst_ref<Timer> data type 44
Figure 5.5: The Microwave Oven external entities 44
Figure 5.6: The Microwave Oven class diagram 45
Figure 5.7: The profiled DoorID attribute in the Door bpClass 46
Figure 5.8: The profiled R4 relationship among Door and Oven bpClasses 46
Figure 5.9: The Microwave Oven system described by the proposed profile 47
Figure 5.10: Microwave Oven's Door xtUML State Machine 48
1 Introduction

Considering 1983, when Antoni Guadi designed one of the first construction models for his futuristic project known as Sagrada Familia, modeling has always played an important role in the development process. Their usage allows to radically break down project costs and developing time, providing also a formal, well-defined model of software systems. In order to provide secure software, the testing phase must be fully completed, but very often it is not. From the industrial point of view, an eventual lack of correctness of systems may strongly affect quality and performances. For these reasons, different tools have been developed to bypass all these problems but most of them are proprietary and model interchanging has become a hard task for engineers.

Nowadays, Model-Driven Engineering (MDE) allows to develop systems both platform-independent and executable. Through software models, validation and verification phases can be strongly improved. Unified Modeling Language (UML) released by the Object Management Group (OMG) in 2001, is a general-purpose language initially defined for requirements analysis and design. Further, its informality may cause different interpretations of the same model. Some improvements have been fixed on its new version (currently 2.3), but some others have also been raised [25].

This thesis work is mainly focused on the definition of a pivot profile, respecting all modeling constraints defined by the fUML and ALF specification, built on the Papyrus framework, an open-source tool whose main goal is to fully replace proprietary tools such as BridgePoint by MentorGraphic or Rational Software Architect by IBM, providing an easy-to-use interface with user-customization facilities.

The Foundational Subset for Executable UML Models (fUML) has been recently released to avoid modeling ambiguities and inconsistencies, achieving the main goal of model execution at the same time. It has become fundamental in model transformation for execution aims, allowing transformation from UML to fUML elements and from fUML to third-generation (3G) languages, such as Java or C++. The usage of a concrete syntax to define both structural and behavioral concepts can significantly improve model execution, allowing behavioral aspects of UML models to be directly parsed and executed by model compilers. Action Language for Foundational UML (ALF) is tailored to the thesis goals and has been used to define the mapping process from the Executable and Translatable UML (xtUML) model to fUML. For this reason, xtUML models have been fully described by the proposed profile, in order to achieve also model transformation from the pivot profile to IBM Rational Software Architect (RSA) meta-model.
1.1 Environment

This Master's Thesis was written during the last year of the double-degree Master program denominated *Global Software Engineering European Master (GSEEM)* promoted by *Mälardalen University*, Västeras (Sweden) and the University of L'Aquila (Italy).

The work was developed at the IDT Department of Mälardalen University in a project supported by Ericsson.

1.2 Objective

Due to the increasing interest of industries on model-driven techniques, model-transformation is starting to play an important role in software development process. The objective of this thesis is to investigate the chance to support models exchanging among three different, widespread, proprietary tools, through horizontal transformations.

1.3 Literature

All references and knowledge used to write this report have been collected by using manuals and the Internet. In addition, face-to-face and remote meetings with thesis supervisors added better quality and information to the final work. For all references to books, papers, articles, and reports used in this work, Bibliography section is provided at the end of the report.

1.4 Convention

Due to different text aspects wrote in the report, different font-styles have been used in this thesis to improve readability. The *Italic* style is used to highlight domain-specific terms used for the first time, while *Courier New* font indicates scripts, parts of code or everything else is strictly related to coding or profiling. Instead, the *Bold* style is used to highlight important concepts in the report.

1.5 Audience

The audiences for this thesis are at least those with an undergraduate degree in Computer Science or
medium knowledge of the Model-Driven Software Engineering, since all main, domain-specific concepts, adopted to explain topics under discussion, are detailed explained along sections.

1.7 Thesis Outline

The report is organized as follows:

**Section 2** presents a brief overview on all technologies and methodologies referred by this work. The Model-Driven Engineering approach and its inheritance concepts are early described. Afterwards, a full description of all the OMG standards used into the thesis work is provided.

**Section 3** describes the involved tools, highlighting their own main differences and providing an evaluation on their advantages and disadvantages. Furthermore, a general description on how these tools are involved in tool chaining process is also described.

**Section 4** presents the proposed profile, According to those standards described in Section 2. Well-known profiling approaches are taken into consideration and implemented over structural and behavioral definition. Moreover, all necessary information to test and deploy the proposed profile onto the Papyrus framework is provided as plug-in extension.

**Section 5** presents the Microwave Oven system, available in BridgePoint distribution [12]. A semantically equivalent system is described by using the proposed profile in collaboration with the ALF action language.

**Chapter 6** reviews the most important concepts stated in this thesis work, evaluating the goal achieved and suggesting future works on the same topic.
2 Background

During last decade, software industry and research environments felt the necessity to radically improve software development and testing phase. Due to bigger and bigger complex systems where thousands of line of code are required, quality, performance, maintainability, interoperability, and portability became the keywords to produce efficient, well-structured software [26].

The following sections provide an overall overview on techniques and approaches used to develop good-quality software, according to worldwide industries demand.

2.1 Model Driven Engineering

*Model-Driven Software Development* (MDSD) helps developing teams to release high-performance, software. Indeed, by using abstract models, hand-made code can be directly generated, independently both from the programming language and the platform used by the real system [26].

![Figure 2.1: The Model-Driven Architect (MDA) approach](image)

In the late 2001, the *Object Management Group* (OMG) [1] released a standard approach for software development, called *Model-Driven Architecture* (MDA) [2]. By using this methodology, domain-related concepts must be defined at the Platform-Independent Model (PIM) level. Furthermore, automated model-transformations translate models from the PIM level to the Platform-Specific Model (PSM) one, where domain-specific concepts are defined at more abstract level, according the entire source system.
Thereafter, the source code for a specific platform, represented at PSM level through easy-to-handle concepts, can then be automatically generated.

In fact, the idea behind the MDA approach is strictly related to the awareness that technologies and platforms, are changing faster than concepts. Using formal transformations, this gap can be reduced, helping systems release and maintainability.

Hence, the MDA approach (Figure 2.1) moves developers focus from coding to the modeling levels, providing a high-level of system abstraction.

Indeed, merging the specified platform models to the common developing phases, the MDA approach can be easily linked to the development software life cycle, as shown in Figure 2.2.

![Figure 2.2: The MDA approach during software development life cycle](image)

An important aspect of MDSD and MDA is *meta-modeling*: it allows to describe the model structure among four definition levels, as standardized by OMG.

Figure 2.3, shows how these levels are related one another. Starting from the bottom part of the figure, the first level is the M0-level where the real system is instantiated. Obviously, the system is conforming to a conceptual model specified at M1-level. The model can be specified in three ways: (1) using the Unified Modeling Language (UML) [3], (2) extending UML (UML Profiling), (3) by means of other modeling languages. Above the M1-level, the meta-model (MM) M2-level defines those concepts used in the Model level (M1). Finally, at the summit of the pyramid, the meta-meta-model M3-level is specified by means of Meta-Object Facility (MOF), the OMG’s meta-meta model used to define modeling languages, related concepts at M1-level, such as UML or xtUML.
Providing a discrete level of granularity, real systems models can be automatically transformed to platform-specific code or to any other semantically equivalent model, simply by using transformation languages such Acceleo [10] or QVT [11], respectively.

![Figure 2.3: The Meta-modelling levels](image)

### 2.2 UML extension mechanism: go heavy or light?

UML is a general-purpose modeling language and for this reason, it does not contain domain-specific concepts. However, its structure provides an important extension mechanism, specifying concepts for specific domains:

- **Heavy-weight** extension, which adds new UML-non-conformant concepts at MOF level (M3). This mechanism causes loosing of tool support usually granted by UML.

- **Light-weight** extension, which extends the UML semantic of defined concepts at model M1-level, by using stereotypes, tagged values and constraints at meta-model M2-level. This mechanism, called Profiling, allows resulting models to conform to UML and hence, be compatible with any tool that support this standard.

According to tools features and the main goal of this work, the latter mechanism has been selected in our work to easily accomplish the tool chaining task, since all the tools under focus support the UML profiling. Another reason since lightweight extension has been chosen on the heavyweight one is related to the behavioral aspects we need to integrate in our pivot profile. Executable UML [5], described in Section 2.4, is compatible with UML through the Foundational Unified Modeling Language (fUML) [6] standard and its concrete syntax, called Action Language for Foundational UML (ALF) [7].
These two ongoing standards (only beta versions have been released) can be applied to any UML model by using simple tricks during model specification, as presented in Section 5. Consequently, behavioral aspects can be added to our class model, allowing execution and anticipating the testing phase.

Summarizing the entire approach used to achieve the goal of this work, four main steps must be fulfilled to build a good UML profile:

- Build the domain model by using relationships and entities like classes, attributes, and constraints;
- Look into UML meta-model and the standard profile to extend domain entities;
- Inherit all that can be inherited;
- Introduce new stereotypes for missing concepts through UML extensions or extension of an existing profile.

By pursuing all of these steps, a new domain-specific profile can be developed and expressive semantic of the model can be extended by adding behavioral aspects to it, enabling a ready-to-execute model.

### 2.3 Object Constraint Language (OCL)

As stated above, one of the main advantages about the lightweight approach is the possibility to easily add constraints into profiles. The *Object Constraint Language* [4] allows developers to add design constraints, defining modeling rules for MOF-based model instances. However, OCL constraints are language-independent: this means that they can be used at any meta-level of the meta-modeling approach.

Some works [8, 9] performed by the CEA LIST’s team, have provided an optimized, automated mechanism to design domain-specific modeling languages by using a lightweight, systematic approach: a UML profile, built on conceptual domain models, which adopts a set of minimal framing rules during concepts definition.

In both works, is shown how using OCL constraints, has a deep impact on optimization if used during profiling to guarantee rules at model level. When a relationship association between extended classes is necessary and the respectively meta-classes are already related each other, a good solution in a well-defined profile requires one or more OCL constraints instead of associations among stereotypes.

Our work follows all advices suggested in [8, 9], defining OCL constraints to optimize the proposed profile. However, some constraints have been specified by using the natural language, providing a general idea about how the constraint must rule the stereotypes.
2.4 Executable UML: xtUML

UML model execution is a new explorative approach used to reduce software development costs and improve time-to-market delivery. By means of this new technique, phases like testing and debugging can be easily performed on the designed model, adding a specific behavior to its elements: Executable UML [5] is a well-defined, UML 2.0 profile that enables this approach. Thanks to its granularity, a PIM can be designed to specify software and hardware systems. Indeed, Executable UML supports the OMG MDA initiative, getting more and more interest from Model-Driven Engineering community. Nowadays, not so many action languages exist. However, UML-based environments like BridgePoint [12] by MentorGraphics or Action Specification Language (ASL) [13] by KennedyCarter provide a complete, high-quality specification of UML action semantic.

Executable and Translatable UML (xtUML) is based on object-oriented approach that accelerates the developing process of real-time embedded and technical software systems [16]. Its main advantage is the complete separation of application and software architecture design, during the development process, as shown in Figure 2.4 [16].

The Application Model contains all concepts needed during system specification, in order to provide an executable model. The Software Architecture, defined by means of design patterns and rules, is completely independent from the application and it is included into the Translator engine. In fact, it generates code for the target system, mapping the Application Model onto the design patterns. Consequently, the target system is conforms to the application model and software architecture design.

Figure 2.4: The xtUML development process
Relating to how xtUML provides model specification and execution, during the “MODPROD Workshop on Model-based Product Development” at the University of Linköping (Sweden) in 2007, Erik Wedin presented a more in-deep explanation about how xtUML can actually help developers during the development phase. In his presentation about “Model-Based Development of Embedded Systems with MDA and xtUML”, he shown Figure 2.5, helping the audience to better understand the main concepts behind xtUML process and how these concepts are related one another, from the requirements analysis to the translation phase. The boxes labeled from one to six, represent those activities related to the Platform Independent Model (PIM) aspects, according to the MDA guidelines. The eighth and ninth box highlights those activities related to the target architecture. Finally, the tenth and eleventh box has been used to bind the system-design activities, such as model transformations and testing.

![Figure 2.5: The xtUML process](image)

According to Figure 2.5 and the xtUML specification, four main models are required to provide model execution by using xtUML:

- **Domain Model**, which describes the interactions between different domains;
- **Class Model**, which describes classes, attributes, relationship and constraint of the structural model;
- **State Model**, which describes states, transitions, asynchronous behavior of the behavior model;
- **Action procedures** specified by using the *Object Action Language* (OAL) [14]. OAL is actually used in the processing phase and executed when a behavioral action is launched.

All concepts defined as active classes in the Class Model have their own State Machine diagram, which is independently executed. Each State Machine state runs one or many actions, synchronizing its behavior with other objects or calling an internal method.

### 2.4.1 The Class Model

xtUML’s classes have similar structure and semantic of standard UML classes. They are represented by rectangle boxes and composed by three main compartments used to specify their own properties, operations, and events.

During property specification, designer can use three kinds of data types: *built-in* (commonly called *primitive*), *user-defined* and *enumerations* data types.

**Built-in data type** indicates those data types defined into BridgePoint. In addition to the standard UML primitive types, such as String, Integer and Boolean, four primitive data types have been specified to extend attributes semantic:

- **Real**, is used to represent real values;

- **unique_id** specifies a unique value across all instances in the system and it is assigned to a primary key attribute.

- **inst<Mapping>** is used when an xtUML model needs to access to a data structure from another xtUML model.

- **inst_ref<Mapping>** has the same meaning of the previous data type but the data structure is passed by reference instead of its entire structure.

**User-defined data type** refers to those data types specified by developers and based on primitive data types, describing domain variables.

Finally, **Enumeration data type** keeps the same meaning held in UML

Once data-types have been fully defined into the system model, xtUML attributes must relate to them. Attributes are described as `<derived><name> : <datatype> {<property_string>}` where `<derived>` indicates whether the attribute is derived ("/") or not. The `<datatype>` entry specifies what primitive type is related to the current attribute and `<property_string>` contains all information to identify either primary/alternative keys or attributes dependent to external relationships. For instance, the property string `{I, I2, R2, R3}` is used to represent the referential
attribute obtained by R2 and R3 relationships, which is either entirely or partially part of the primary and alternative key of its class.

An additional feature about relationship is obtained by constrained relationship, shown with the ‘c’ character after the association name (e.g. R2c). In [5], Shlaer and Mellor provided a useful example to understand how it actually works. Figure 2.6 [5], shows a constrained relationship between students and teachers when both join the same department.

Figure 2.6: Example: student advised by a teacher from his same department

An important semantic distinction between UML and xtUML, is related to the operation concepts specified into classes. In fact, it is common to find an xtUML Class Diagram specified in the system model without any operations into its classes. Since Class and State Diagram models are related each other, class behavior is directly described through state machines, associated to each class. In fact, as explained in Section 2.4.2, designers can use two different ways to specify state machines by using OAL: Instance State Machines or Class State Machines.

Regarding relationship association, xtUML does not use the complete association set specified by UML specification to connect classes. In particular, the only kinds of relationships that can be used in the xtUML model are generalization and association, with restricted expressive power. In fact, the only cardinality that must be used by any xtUML associations is the one-to-one, one-to-many, zero-to-many, or zero-to-one multiplicity.

Similarly, xtUML generalization-set association has restricted semantic since the disjoint and complete property ({disjoint, complete}) must be set to true, forcing to instantiate an unique object to any sub-class of a super-class [3,7]. An additional important property is regarding the super-class involved
in a generalization association. In fact, any super-class must be specified as abstract class, as required by xtUML in its execution methodology at modeling level.

Finally, regarding operations, they are described in BridgePoint as $\text{name} (\text{arguments}) : \text{returned\_type}$, similar to a common programming language like Java/C++.

A practical example of an xtUML Class Diagram is shown in Figure 5.6 where the class model of the common Microwave Oven system is presented.

### 2.4.2 The State Model

Different to Foundational UML (fUML), presented later on this report, xtUML expresses behavioral aspects through the State Machine model. It has no particular semantic and syntactic restrictions from the standard UML State Machine model: states, events, transitions, and activities might be used to define behavior of classes’ operations. The only advice suggested by the xtUML specification, is to keep the State Model as more as clear and simple, avoiding states nesting.

However, an important distinction on state machine type must be underlined. If a State Machine of an active class is specialized as an Instance State Machine, actions have to be provided per each instance of the class. Differently, if it is specialized as Class State Machine, actions define the behavior for the entire class. Usually, an Instance State Machine is the common specialization used to describe behavior for a class. However, a concrete syntax language that match with the action semantic specified by the state machine model, needs to be provided to execute the model. xtUML uses OAL to define the action semantic for any UML model or, in simple words, to define the semantic that occurs in an action [14].

Focusing on the language, OAL provides five types of action process: data accesses, event generation, test, transformation, bridge, function, and inter-component messaging. When the execution starts, the state machines run concurrently, changing the state when an event happens. OAL manages the processing during every action providing rules and all the necessary constructs used to simulate the UML models.

A practical example of an xtUML State Machine, is shown in Figure 5.9 where the State Machine model of the Microwave Oven’s Door class is presented.

### 2.5 Foundational UML

Since 2001, UML behavior can be specified by means of concepts and actions contained in UML activities, describing model’s behavior by using flow of data, tokens, and control nodes. Unfortunately, the semantic associated to these concepts can be informally described when UML is used as modeling...
language. Due to this high-level of informality, ambiguity, and incongruence might be found in the semantic description of the activity diagrams and in other behavioral models [27]. Some improvements have been made in version 2.0 of the UML Specification, where part of these problems has been fixed, but some others raised [25]. With the aim to avoid all these problems, several ideas on new action languages standard have been proposed during the previous years and new formal languages have been released for domain-specific aims.

In October 2010, OMG standardized a new action language, called *Action Language for Foundational UML Models* (ALF). It is strongly based on the *Foundational Subset for Executable UML* (fUML) [6] and providing a concrete syntax to its models, as described in Section 2.6

Nevertheless, what is exactly fUML? In 2005, OMG released a *Request for Proposal* (RFP) [19] document to specify a subset of UML 2.0, with the aim to define its execution semantics. In fact, fUML specifies an abstract execution semantic for UML activity diagrams, enabling formal models execution, through the Base UML subset. Furthermore, a direct translation from UML to fUML concepts and from fUML to a common object-oriented language platform like Java/C++, can be easily accomplished. Figure 2.7 [6], shows how the fUML subset can be used as an intermediary step during the transformation from a UML model to a computational platform language.

![Figure 2.7: Translation to and from fUML](image)

As already state above, the fUML meta-model is made by a sub-set of UML package. The fUML specification, based on UML 2.3, highlights three main packages to relate structural and behavioral aspects to each other: the UML2 packages, the *Execution Model* and the *Model Library*, all described by its meta-model.

The UML2 packages represent the *conformance level* that corresponds to the UML *compliance levels*. Figure 2.8 [20], is based on Figures 2.2, 2.3 and 2.4 of the UML Superstructure specification [3]. The
compliance levels L1, L2 and L3 are shown below the UML concepts on the bottom part of the figure. As described in [3], at L1-level the BasicBehaviors package is merged, encapsulating behavior into classes operations. At L2-level, the Profile package is merged, encapsulating behavior into stereotypes operations. Finally, at L3-level, the templates package is merged, allowing classifiers and packages defined with a L3-compliant tool to have template signature [20]. Therefore, an UML model able to respect all the necessary constraints, is easily represented by fUML, describing both static and behavior support.

Figure 2.8: UML Compliance Levels

The Execution Model package represents the operational specification of its execution semantic in a behavioral context, such as UML Activity. In this case, the specification of activity elements is expressed by using the Base UML (bUML) subset, the first-order logic axioms used to provide base semantic to the execution model of fUML. An important sub-package of the Execution Model is the Loci package: it represents the abstract specification for fUML execution engines and it specifies the execution environment to run the models providing abstract internal interface of the execution model [6].

The last package of fUML is Model Library, which contains execution semantic of the set of primitive functions, available in the execution context. However, the package has to be used only when OpaqueBehavior or FunctionBehavior have not been specified to the element under focus.

Finally, an important aspect of fUML is related to its structural model, due to some important changes from UML Specification. Packages like Association Classes, Dependencies, Interfaces, and Power-Types defined in UML have been excluded, or partially excluded, from fUML Specification, together with some other important features. In fact, as described at the beginning of this section, the idea behind fUML is to release a formal version of the UML specification, where ambiguities and inconsistencies have been left out from the designed model.
2.6 UML Action Language: ALF

Three years later the fUML RFP document, the OMG community released a new proposal [7] to standardize the action semantics of a UML-designed model, providing an executable model independent from the target platform and its related language.

Through UML action language, designers are able to create a PIM-level model to specify model behavior by means of an easy-to-understand language, satisfying testing, debugging, and simulating goals, at the same time. For instance, during the design phase in Rational Software Architect (RSA), is often required to integrate additional features to the model elements of the system under study. To satisfy this task, marking models have to be specified through the UML Action Language (UAL), helping designers during model-to-code (M2C) transformation, bridging concepts to the domain-specific target language. Subsequently, once the model is created at the Platform-Specific level, the transformation to the specific 3G language may then be actuated.

Hence, UML action language is an easy-to-use and a well-defined action language that can be used to specify the action behavior of a UML model. Indeed, highly trained developers are not required since UML has become a widely used modeling language and intermediate-skilled designers can actually work on the system.

Similar to UAL, fUML is nowadays a complete set for behavioral system models specification. In October 2010 the Action Language for Foundational UML (ALF) [21] standard has been released by OMG, two years later Request-for-Proposal [7].

ALF defines a concrete syntax to a UML behavioral model, mapping its concrete syntax to the fUML abstract syntax, providing execution semantic by means of fUML. Indeed, "the result of executing an Alf input text is thus given by the semantics of the fUML model to which it is mapped, as defined in the fUML specification", as stated in [21]. The ALF action language is based on UML 2.4 because this UML version includes some corrections on abstract syntax constraints, necessary for static semantic analysis of ALF. Differently from ALF, despite fUML is based on UML 2.3, there are no consistent structural modifications in UML 2.4 abstract syntax and for this reason, the matching is perfectly achieved [21].

Looking at the ALF concrete syntax, it is a Java-like language, able to define textual syntax during structural modeling. One of the main advantages is strongly related to its execution semantic. Indeed, the modeling tool can executes the ALF text body in three ways:

- It can be directly executed by the execution tool
- It can be compiled to a fUML model and executed, respecting the semantic specified by fUML
- It can be translated into some executable form outside the UML platforms and then executed
The main goal of ALF is providing alternative ways to specify model behavior in a traditional UML model, simply by using its concrete syntax elements, as shown in Figure 2.9 [21]. Every time is possible to add *Opaque Expression* in a UML model or activity model, an ALF *Expression* may be used as text body instead. Alternately, ALF *Statements* may define the text body of *Opaque Actions* of UML Structure Activity-Node. Similarly, it may also be used as body of an *Opaque Behavior* or compiled into a UML Activity [21].

Finally, ALF also adds the *Units* concept, extending the standard UML package and namespace concepts. The ALF *Unit* is defined as “*a namespace defined using Alf notation that is not itself textually contained in any other ALF namespace definition.*” [21]. Thus, an ALF *Model Unit* can be used to refer to external UML models, referenced as name element, such as External Entities in xtUML.

![Figure 2.9: The ALF Syntax Elements](image)

Hence, ALF may then be adopted by designers to achieve those tasks, such as behavior specification through a 3G-like language, raised by the meta-modeling approach during the model system specification. Furthermore, it can be parsed and then executed by the fUML once the latter will be finally released. Meanwhile, R&D companies sectors and research activities are analyzing ALF specification, in order to provide a compliant mapping to proprietary tools.
3 Tools differences

In Section 4, we present our pivot profile developed on the Papyrus [18] platform, aiming to reflect models semantic, expressed by BridgePoint [12] and IBM Rational Software Architect [17].

In the following subsections, an overview about the advantages and disadvantages of the involved platforms is provided and a final discussion on the main differences, provided.

3.1 BridgePoint

Built by MentorGraphics [22], BridgePoint is one the most advanced Platform-independent, UML-based environment adopted by many big companies to improve the software developing and testing phase. Thanks to its high-quality and code conformation to any convention or standard, structural and behavioral formal models can be easily designed, and optimized domain-specific code, executed. Figure 3.1 [22], shows a general overview on how the BridgePoint framework actually works.

![Figure 3.1: The BridgePoint development process](image)

The structure of a generic system model is developed by means of the xtUML modeling language, fully described in Section 2.4. The development focus is then shifted from coding by using common 3G languages, like C or Java, to something largely different, such as abstract models. Indeed, the xtUML
abstract language helps developers to cut lines of code that are usually defined to specify a generic software product. Replacing this activity by a model-to-text transformation, written by using a transformation language (such as Acceleo [10]) analogue software systems can be developed.

Furthermore, graphical models are well interpreted by and integrated with the system platform, providing user-friendly interfaces that help designers to better specify systems, with a powerful optimized tool.

3.2 Rational Software Architect

Rational Software Architect is an Eclipse-based, UML-compliant tool, powered by the Rational Division of IBM [17]. It largely shares the same goals of BridgePoint, such as the automated development approach where UML models or Domain Specific Modeling Languages (DSL), are commonly used to reduce the risk of failure.

![Figure 3.2: IBM Rational Software Architect framework](image)

However, differently from BridgePoint, it already has some internal specifications for model transformations. In fact, one of the RSA's main features is the possibility to transform UML models into C++/Java code, Common Object Request Broker Architecture (CORBA) Interface Definition Language
(IDL) or other widely used model templates. Therefore, the transformation mapping can be achieved in both ways, enabling tools and languages interoperability.

Another important feature is related to project's team communication, where "requirements and capabilities for multiple target deployment environments (integration testing, performance testing, staging, and production)" [17] are correlated one another, improving quality of the development process and avoiding communication misunderstandings.

Similar to BridgePoint, model execution and simulation is also one of the key features of the framework. Specifically, simulation phase can be launched both to informal and formal models, where UML Action Language (UAL) is commonly used to describe behavioral aspects.

Figure 3.2 [17], shows all main features which surround the Rational Software Architect modeling tool.

### 3.3 Papyrus

Papyrus is a recently released, open-source, UML2 modeling platform based on the Eclipse environment. It is a general-purpose graphical editing tool since it can be easily customized, providing a complete support on UML profiling and model-based system engineering. Indeed, the main goal of Papyrus project is to release a strong alternative to proprietary tools, such as BridgePoint or Rational Software Architect. More and more big companies, like Ericsson, are putting their effort on the specification of an open-source, Eclipse-based modeling framework able to design critical systems and allow model execution. To achieve this task, the Papyrus framework is based on the common Eclipse modeling components, such as the Eclipse Modeling Framework (EMF), the Graphical Modeling Framework (GMF) and the Graphical Editor Framework (GEF) [23].

UML 2.0, is the commonly used language during model design. However, Papyrus provides extension mechanisms to add modeling languages, such as MARTE or other DSLs, or customized features through extension points, allowing a well-structured extensible tool.

From graphical point of view, Papyrus Diagram Editors (PDE) refers to all those standard editors specified on Papyrus for UML and SysUML models.

Figure 3.3 [18], shows the Papyrus's structural components based on the Eclipse framework, as previously described.
3.4 An Evaluation

Both BridgePoint and IBM Rational Software Architect use class diagrams and state machines to describe static and behavioral aspects. Differently, the new idea behind Papyrus is weaving these aspects to a canonical model, where the behavioral concepts have been directly integrated into class diagrams, the only kind of diagrams used during the system specification. As presented later on this report, a practical example would be to build a canonical model with only static and behavioral concepts provided by the target programming language at meta-modeling level (M2), generating domain-specific code (M0) once the system model has been fully described at model level (M1).

The main advantage on following this idea is related to the separation of modeling concerns (such as UML, fUML, and code), improving maintainability, and extensibility of system models during explicit mappings. On the other hand, the main disadvantage concerning this belief is due to the necessity of two model transformations to convert traditional models (e.g. those described by using both Class and State Machine Diagrams) to canonical models (described only by using abstracted concepts of the target 3G language). The former M2M transformation is forced to weave both structural and behavioral aspects in a single fUML model (without State Machines). The latter M2C transformation, instead, is actuated to convert the generated canonical model, to the target code. Hence, even if this task requires a small extra effort to be accomplished it does not affect the main advantages obtained by Papyrus.

Unfortunately, model execution and simulation is not available yet. However, as previously claimed, many companies and academic research groups are putting their effort to achieve this task. Thanks to the Papyrus project, we believe they will be achieved in the near future, thanks to the Foundational
UML (fUML) UML subset, and the Action Language for Foundational UML (ALF). In fact, as already claimed in Section 2, fUML also specifies the Base UML (bUML) subset that contains the first-order logic axioms used to provide base semantic for the execution model.

Looking at tools main features, an important role is then played by tool chaining approach. Since many industrial companies use different tools for different development phases, tool chaining must be available between two or more platforms. Tool functionalities change rapidly and keeping trace of each modification in a software application is not a trivial task. This can have a deep impact on those concepts that, defined by means of older versions of the same tool, might be not fully represented in newer versions due to language core modification e.g. modification to the meta-model. Another important issue is related to information exchange, since different tools require different formats and a standard format has not been released yet. Horizontal transformations and bridge operations at meta-modeling level must also be able to keep trace of differences between the involved meta-models, providing a unique conversion. Unfortunately, most of the advanced modeling, tools are proprietary and companies usually prefer them to the open-source ones, due to maintenance and tool support issues. However, when tool chaining is required, developers do not have full access to the modeling concepts behind the tool due to proprietary, business restrictions. The interchanged model and its semantics can then be partially invalidated, causing an incomplete translation of concepts between two different tools through model-to-model (M2M) transformation.
4  Tool Chaining: the pivot profile

As mentioned in Section 2.4, Executable and Translatable UML is nowadays commonly used to design models with action semantics. Unfortunately, it is a proprietary language and developers cannot hold a complete control on both structural and behavioral model aspects. If the available elements are not enough expressive to describe necessary concepts, they must be specialized in a way that might be semantically different to the intended ones. Having this kind of lack of control during development, might have a deep impact on the system model design. However, automatic conversation model-to-code or model-to-model may significantly decrease the risk of project failures at developing time. Therefore, a certain level of formality must be provided when a preliminary testing phase is executed directly on the model.

An open-source pivot language would significantly decrease the communication efforts between different proprietary tools, allowing model interchanging, and providing a certain level of system model abstraction. Furthermore, it enforces the model specification to be semantically compliant to a well-specified subset of modeling standards, using a language defined as “lingua franca” for the involved tools.

In the previous sections, we defined fUML as a subset of the UML specification where not all UML elements must be adopted to specify its models. For this reason, during the case study development, some tricks must be pursued to create a model conforming to both the fUML and xtUML specification. Furthermore, some elements semantically equivalent to the UML AssociationClass, Interface, PowerType, and Dependency concept, cannot be used in a fUML model. Hence, model designers must be able to draw the system model only by using packages, classes, properties, associations, and operations. Instead, system behavior must be defined only through actions and all related concepts have to be expressed by an UML Action Language (UAL), such as ALF. In fact, according to the idea behind the Papyrus project, both the static and behavioral system aspects must be described by one, canonical, class model.

The following sections present main concepts of the pivot language and how these have been defined, according to the fUML and xtUML specification, as well as the standards supported by RSA. Since the access to the proprietary tools has been limited by knowledge, proprietary restrictions, some other primitive types might take part in the xtUML specification.

4.1  Primitive Types

According to Section 2.4, four new primitive types have been added to the proposed profile (Figure 4.1):
- **Real** is used to represent numbers with the fractional part: e.g. 1.0, 2.5 and so on. Similar to the standard, UML Integer primitive type, arithmetic, unary, and comparisons operations have been provided.

- **unique_id** is used to represent unique instance values in the system. Two comparisons operations are available for this primitive type: ==, !=.

![Figure 4.1: Profile’s Primitive Types](image)

- **inst_ref<Mapping>** is used to represent data structures passed by reference from another xtUML model. Two comparisons operations are available for this primitive type: ==, !=.

- **inst<Mapping>** is used to represent the data structures passed to the system from other xtUML model. All standard comparisons operations are available to this primitive type: ==, !=, >=, <=, >, <.

### 4.2 Structural concepts

Similar to Papyrus, RSA supports UML 2.2 to specify system models, as presented in Section 3.2. Thanks to this important feature, the mapping process from RSA models to the profiled ones may be easily accomplished. Unfortunately, some limits have been encountered during the pivot specification, as presented later on this section.

Since several concepts cannot be used in a fUML system model, some stratagems must be pursued during model chaining, leading the RSA models to conform to the pivot profile and vice versa. For instance, an abstract class associated to a certain class entity has to be used in the pivot model to represent the RSA Interface concept. Analogously to BridgePoint models, Association Classes can also be used in the RSA ones: for this reason, the same stratagem adopted for xtUML models has been used to map the pivot profile to RSA, as presented in Figure 4.6.
Unfortunately, sometimes seems hard to fulfill the mapping process. The following subsections present the structural elements in the pivot language. Performing this task a well-defined set of stereotypes have been defined as extensions of specific UML meta-classes in order to provide both xtUML- and fUML- compliant concepts, enabling tool chaining among the involved tools.

4.2.1 The bpSubsystem stereotype

Each xtUML package represents a specific sub-domain of the system project. According to the xtUML specification, four different types of xtUML package have been specified (see Figure 4.2).

Profiled packages, such as bpDomainPKG, bpExternalEntityPKG and bpDatatypePKG do not provide any other additional information to the standard UML Package meta-class.

Differently, the bpSubsystemPKG stereotype provides additional information, such as rangeFrom and rangeTo, which respectively specify minimum and maximum identification number, used by the classes into the package. The keyLetter tagged value uniquely identifies the subsystem through a string value. It is also used in conjunction with class’s keyLetter to uniquely indentify a class inside each subsystem.

![Figure 4.2: The abstract bpPackage stereotype](image)

4.2.2 The bpDatatypes stereotype

In addition to the primitive types, user data types may be added, at model level, by using the
bpDatatypes stereotype, extending the UML Datatype meta-class. The referred primitive type must be specified into the referredPT attribute during propriety specification.

However, Datatypes cannot have OwnedOperations, which means, “[…] there is no way to provide executable methods for the operations of a data type” [6] specified in RSA models. In this case, loss of information might cause problems during model transformation that might be solved by developer adding extra information during the transformation specification.

4.2.3 The bpClass stereotype

The bpClass stereotype (Figure 2.3) is one of the most important concepts in our profile, since only classes have to be adopted to represent system’s elements. Since each class may represent a different element in the real system, this information needs to be specified in the bpClassPurpose enumeration attribute.

Each class contains bpAttributes and bpOperations, specified in the attribute and operation compartment, respectively. Classes are related each other through relationship associations, stereotyped as bpRelationship. All these inherent concepts are presented later on this report.

To enforce these rules, two OCL constraints have been defined into the involved stereotypes. The OCL constraint related to bpAttribute has been defined as follows.

```oclm
definition self.base_Class.getAllAttributes() =>forall(p:Property | p.getAppliedStereotype('Profile::bpAttribute') <> null)
```

The constraint presented above states that all attributes of stereotyped classes must be stereotyped as bpAttribute. Analogously, an OCL constraint has been specified to force the use of bpOperation operations, into the stereotyped classes. Obviously, only bpClass stereotypes must be used in the system to specify the system’s classes.

According to the xtUML class specification, additional tagged values in the bpClass stereotype have been used to specify the class’s property string (shown below), the class name, and bounded by curly brackets ({{property_string}}) in any xtUML model. The property string has been specified in the pivot profile through the number and key_letter tagged values.

Finally, the purpose property has been used to provide additional information about the aim of the class, According to the class scope description, given in [5].
4.2.4 The bpExtEntity stereotype

An important aspect of a xtUML model is defined by the External Entity (EE) concept. All those necessary elements defined out of the class model as xtUML models or hand-made code has to be represented by using the bpExtEntity stereotype in our profile. In fact, similar to the bpClass stereotype, it extends the UML Class meta-class but, differently, it does not provide any other information in addition to its name.

However, the system model for external entity specifies only its operations during the modeling phase, which must be stereotyped as bpBridgeOperation operations by the profile.

4.2.5 The bpAttribute stereotype

The abstract, bpPropertyString (Figure 4.4) stereotype extends the UML Property meta-class, identifying those attributes specified into the bpClass elements during the model specification.

All class’s attributes carry more information than standard UML properties. Additional information to identify both primary (e.g. \{I\}) and alternative keys (e.g. \{I2, I3\}), have been used during property specification. Consequently, the bpPrimaryKey and bpAlternativeKey stereotypes describe the primary- or alternative-key attribute of a certain class, respectively.

Furthermore, attributes may be specialized by the bpReferentialAttribute to describe all those attributes owned by other classes in the model but also used into the current class to identify external attributes, similarly to the external key concept defined in Entity-Relationship (ER) models. The referential mode, identified by refMode property, must be set choosing among localRoot and referredToRoot enumeration value to indicate how to build the name of the referential attribute.
As stated in Section 4.2.3, all stereotyped bpAttribute must be owned by classes stereotyped as bpClass. The following OCL constraint satisfies this rule:

```ocl
def self.base_Property.owner.getAppliedStereotype('Profile::bpClass')<>null
```

Furthermore, since the referential attribute name might be automatically generated from the referred attribute selecting the referredToRoot entry in the bpRefAttributeMode enumeration list, an additional constraint need to be specified at meta-model level.

![Figure 4.4: The abstract bpPropertyString stereotype and its generalization](image)

**Figure 4.4** The abstract bpPropertyString stereotype and its generalization

### 4.2.6 The bpRelationship stereotype

The **bpRelationship** stereotype extends the UML Relationship meta-class, adding the number tagged value to its properties, uniquely identifying relationships association in the system model.

According to the xtUML specification, only four types of values have to be used during system modeling to indicate association’s cardinality, as expressed in Figure 4.5. In order to enforce these rules, the RelationshipCardinality and ConditionSide enumeration types define association cardinality (one_many, many_many, one_one) and condition-side (left, right, both, none) during relationships specification, respectively.

For example, an associations with the one_many value in RelationshipCardinality property and condition side left, represent the common multiplicity 0..1 : 1..*. Hence, the card and condSide tagged values have been added to the bpRelationship stereotype, defining all cardinality combinations, as summarized by Table 4.1.
<table>
<thead>
<tr>
<th>Cardinality</th>
<th>Constraint side</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>one-to-one</td>
<td>One</td>
<td>1 : 0..1</td>
</tr>
<tr>
<td>one-to-one</td>
<td>Both</td>
<td>0..1 : 0..1</td>
</tr>
<tr>
<td>one-to-many</td>
<td>One</td>
<td>0..1 : 1..*</td>
</tr>
<tr>
<td>one-to-many</td>
<td>Many</td>
<td>1 : 0..*</td>
</tr>
<tr>
<td>one-to-many</td>
<td>Both</td>
<td>0..1 : 0..*</td>
</tr>
<tr>
<td>many-to-many</td>
<td>One</td>
<td>1..* : 0..*</td>
</tr>
<tr>
<td>many-to-many</td>
<td>Both</td>
<td>0..* : 0..*</td>
</tr>
</tbody>
</table>

*Table 4.1: bpRelationship cardinality combination*

One of the main problems that may be encountered during model transformation is related to relationship multiplicity. As stated before, not all multiplicity combination can be adopted in xtUML models. Hence, during the mapping process from the RSA to the profiled models, loss of information might cause mismatching problems between these models.

Furthermore, associations in the proposed profile may be specialized as `bpReflexAssoc` or `bpConsRelationship` association. The former is used to represent *auto-reference* associations with instances in the same class of the association starting class: in this case, the read-only type tagged value has been set to ‘acyclic’. The latter, instead, is used to represent constrained associations, defining the relationship association as part of a constrained loop.

In fact, an important role is played by the `bpRelationship` stereotype in the mapping process from the (xt)UML `Association Class` concept to a new, semantically-equivalent concept in our profile. As already discussed, association classes cannot be used in fUML modeling. How is it possible to replace this element, then? This task is easily achieved defining `bpClass` elements and `bpConsRelationship` associations set to the right multiplicity values. Subsequently, the relations lists all constrained associations involved in the loop, mapping `AssociationClasses` to standard classes and associations, at model level. Figure 4.6 shows how to replace the (xt)UML Association Class concept during model specification.

*Figure 4.5, shows the defined bpRelationship stereotype and its generalization to bpReflexAssoc and bpConsRelationship stereotypes.*
Figure 4.5: The bpRelationship stereotype

Figure 4.6: How to replace the AssociationClass concept in the profiled model
4.2.7 The bpGeneralization stereotype

An important aspect is related to fUML PowerTypes elements and, in particular, to the GeneralizationSet relationship association. In fact, according to the fUML specification, PowerTypes and GeneralizationSets are not considered fundamental at modeling level, since they "...add significant complexity to the semantics of generalization, particularly as it relates to typing and polymorphic operation dispatching" [6]. The (xt)UML GeneralizationSet concept must then be mapped to common generalization associations. This means that a good mapping from the RSA to the pivot language have to automatically translate all occurrences of GeneralizationSet elements in RSA models to Generalization associations in the profiled model.

However, the xtUML GeneralizationSet relationship association needs to satisfy the ‘disjoint’ and ‘complete’ UML properties in any xtUML model, as presented in Section 2.4. Hence, all sub-classes defined by xtUML, do not have common instances and "every instance of a particular general classifier is also an instance of at least one of its specific classifiers for the GeneralizationSet" [3]. In addition, every generalization has its own identifier number, similarly to any other system association. Figure 4.7, shows the proposed stereotype that represents the Generalization concepts as described by the xtUML specification, extending the UML Generalization meta-class.

According to the modeling rules discussed above, two constraints have been also specified by using the natural language:

- All classes involved (super- and sub- classes) in the generalization association must be stereotyped as bpClass.
- isDisjoint and isComplete properties available in the UML Generalization meta-class, must be set to true.
4.2.8 The bpOperation stereotype

Each operation in xtUML have to be specified as instance-based or class-based operation, setting the scope tagged value of the bpOperation stereotype, with the right entry in the enumeration list. This categorization, may be applied at model level, since how the operation is effectively implemented in the real system, is related to concrete syntax of the action Language.

The bpOperation stereotype has also been generalized by the bpBridgeOperation stereotype used to represent external entity's operations, discussed in Section 4.2.4. It has the same features of a simple bpOperation element without any other tagged value defined in it.

Finally, bpOperations and bpBridgeOperations stereotypes must be applied only to classes stereotyped as bpClass and bpExtEntity respectively: both constraints are provided in the profile by means of the natural language.

4.2.9 The bpParameter stereotype

The bpParameter stereotype provides extra information to the UML Parameter meta-class. xtUML needs additional features, such as parameter passing mode (by value or by reference), in order to address the concrete syntax of the action language (ALF) during operation specification.

Finally, a constraint has also been specified to guarantee that all bpParameter-stereotyped parameters are owned by bpOperation operations.
Figure 4.8 shows how the UML Parameter meta-class has been stereotyped together with its constraint.

![Diagram showing the bpParameter stereotype](image)

*Figure 4.8: The bpParameter stereotype*

### 4.3 Behavioral concepts

Thanks to the Action Language for Foundational UML (ALF) described in Section 2.6, a concrete syntax can be directly adopted to define necessary behavioral concepts during tool chaining. As Section 5 describes, the concepts mapping from the proposed pivot profile to both BridgePoint and RSA models can be successfully achieved. Horizontal transformations at meta-modeling level from ALF to fUML abstract syntax are also provided, transforming "root objects from the Alf abstract syntax representation to UML elements" [21]. Hence, ALF can be used in the State Machine diagram context, integrating ALF text body into their elements. Nevertheless, the suggestions proposed in [21] have not been already standardized due to some mismatching problems between UAL and ALF specification. However, in the case study presented in Section 5, ALF is able to map the node elements defined in the xtUML State Machine diagram, where the action body is specified by using the Object Action Language (OAL). Similarly, RSA supports UAL as modeling language to design both language and tool independent, executable models. Hence, a direct mapping among behavioral concepts from the RSA to the profiled system model can then be easily accomplished, by using the proposed profile in combination with the fUML and the ALF concrete syntax.

Nevertheless, in-deep research activities are still trying to discover whether the mapping process from ALF to UAL can be fully provided, enabling a standardized approach to translate xtUML models to the RSA ones.
4.4 Embed the profile into plug-in

In Section 2 has been presented how Papyrus is an easy-to-extend platform by means of its Plug-in Development Environment (PDE). The extensibility feature, provided by any Eclipse-based tools, is satisfied by using *extension-points* and *(plug-ins)* *extensions*. Extension-points represent already defined, tool's join points that can be used by the *extensions*, as access point to connect to the framework. Instead, a *(plug-in)* *extension* is a small unit that can be developed separately and shared by in the Eclipse-based frameworks through extension points. Both extension points and extensions must be specified into the *plugin.xml* file in the Eclipse-based tool under focus.

In order to facilitate profile's usability during system modeling phase, the proposed profile has been embedded into a new Papyrus/Eclipse-compliant plug-in. Before the proposed case study has been designed by means of the new pivot profile, few previous steps needed to be followed to connect the Papyrus environment with the developed extensions. In fact, once the *manifest.mf* file has been successfully set up and customized palette toolbar extension (see Annex A) added to the pivot profile package, a new instance of Papyrus workbench has been launched.
5 A Case Study: the Microwave Oven

In this section the Microwave-Oven system, available into the *BridgePoint UML Suite Help*, has been taken as a practical example to verify the correctness and completeness of the proposed profile. The originally static concepts of the system are described by means of xtUML and the behavioral aspects through State Machines where its action semantics defined by OAL.

Integrating the pivot language with the new ALF action language an analogue model is generated on the new Papyrus platform, providing semantically equivalent modeling elements to its xtUML version.

A new instance of the platform is then executed with the defined plug-in embedded to the environment, once the Papyrus framework successfully validates the profile. Subsequently, the profile plug-in is available into the registered profiles list when a new papyrus model is created, together with the customized palette (Appendix A). Indeed, in the palette toolbar available at the right side of the workbench, the profile’s concepts are linked to graphical entries. Dragging and dropping menu elements onto the *workbench*, the profile concepts of the system model are directly created both in *Diagram Interchange* and *XMI* format.

Figure 5.1 shows the profile plug-in applied to Microwave Oven project and the customized toolbar available in the palette toolbar.

*Figure 5.1: The profiled workbench*
Unfortunately, some issues are still unresolved in the new Papyrus version available into the Eclipse Helios distribution when the profile plug-in is integrated to a new workbench instance. In fact, the graphical editor view still does not work properly due to some bugs in the new version of the Java Runtime Environment (JRE), as discussed on the Eclipse Community Forum [24].

However, has been possible to check the correctness of the pivot profile in the same workspace where it has been developed.

According to the Microwave Oven example description, its xtUML model is organized in the following packages:

- **Domain Package** is used to organize all system's packages of the model.
- **Data type Package** is used to describe data types, based on the primitive data type.
- **Function Package** is used to organize functions in the BridgePoint model through the Model Explorer view in Bridgepoint. It is tool-specific feature and there is not an UML notation for them. The developed profile does not define this package.
- **External Entity Package** is used to organize external models or pieces of hand-written code specified outside of the current model.
- **Subsystem Package** is used to organize classes and associations in the Microwave-Oven System
- **State Machine Package** is used to organize the states, events, transitions, and activities to active classes. The developed profile does not define this package because class behavior has been specified by using the ALF action language.

In the following sections, all the necessary packages have been fully described together with their profiled concepts. Therefore, a complete description of the example has been provided at the end of the section.

### 5.1 Domain Package Diagram

Similar to the xtUML Microwave Oven example, the Domain Package Diagram shows the all packages involved in the system specification.
However, differently from the BridgePoint workbench, Papyrus do not provided a direct, graphical relationship between packages and diagrams. However, Figure 5.2 shows how they may be organized: modeling elements can be added into each package and refers to each other independently. The provided stereotypes have been used to specify every package, adding information described in Section 4.

### 5.2 Data type Package Diagram

Data type and Enumeration elements have been mapped in our profile through the elements described in Figure 5.3.

Three user-defined data types have been specified by using the stereotype `bpDatatype` adding more information to the standard UML `Datatype` concept, such as the referred primitive type, as shown in Figure 5.4. Therefore, an enumeration element type has been specified by using the UML `Enumeration` concept in order to define the tube wattages levels of the oven.
Different to the xtUML Microwave Oven model, primitive types have not been specified. Indeed, they are already integrated in the proposed profile and can be directly referred by refDatatype property into any bpDatatype-stereotyped elements.

5.3 External Entity Package Diagram

xtUML External Entities concepts have been added to the model by using the bpExtEntity stereotype, as shown in Figure 5.5.

The microwave oven has three external entities:

- **Control Panel**, which represents the microwave control panel, designed outside of the current domain.
- **Architecture**, which represents the model compiler, according to the example definition. It also contains a bridge operation to shut down the microwave application.
- **Time**, which provides the bridge operations to access to date, time and timer event.

xtUML External Entities are strictly related to the testing phase, performed into the BridgePoint environment. Unfortunately, fUML and ALF are still ongoing specification and an effective test on bridge operations cannot be performed yet. However, we believe this goal will be easily as soon as the two standards will be finally released.
5.4 Microwave Class Diagram

The class diagram of the xtUML Microwave Oven system model is shown in Figure 5.6.

![Microwave Class Diagram](image)

Figure 5.6: The Microwave Oven class diagram

However, the Oven class entity will be the only concept fully describe, since it contains most of the stereotyped elements previously described.

The Class element is specialized by using the bpClass stereotype, providing additional information on its structure, such as the reference number and the key letter, both used to uniquely identify the class in the system. According to the example description, these two tagged-values have been respectively set to ‘1’ and ‘MO_O’. All attributes that require the property string information may be composed by one or more stereotypes that extend the UML Property meta-class: bpprimaryKey, bpAlternativeKey or bpReferentialAttribute stereotype. In this case, the DoorID attribute represents the class’s primary key and only the bpPrimaryKey stereotype has been applied to the attribute.
Differently, the referred attributes (TubeID, LightID, BeeperID, DoorID, and TurntableID) specified by the `bpRefAttribute` stereotype, specify external attribute necessary in the current class. They need both the inherent association among involved classes and the referred attribute in the target class, commonly its primary key. This information has been set by using the `ref` property, as shown by the DoorID attribute of the Oven class in Figure 5.7.

![Figure 5.7: The profiled DoorID attribute in the Door bpClass](image1)

Not all other attributes, require additional information, since they have been defined as standard UML properties.

Lastly, every association in the class diagram has been specialized by the `bpRelationship` stereotype, as shown by the R4 association in Figure 5.8.

![Figure 5.8: The profiled R4 relationship among Door and Oven bpClasses](image2)

As stated at the beginning of this section, the proposed case study has been mainly focused on the Door class specification. Figure 5.9, shows the complete Microwave Oven Class Diagram designed by using the proposed profile.
Figure 5.9: The Microwave Oven system described by the proposed profile
5.5 Replacing State Machines

ALF action language is used to describe the behavioral aspect of the active classes specified in the class diagram, in order to replace any State Machine diagram.

In our case, behavioral aspects have been defined only to the Door class. Two operations, specialized by using the bpOperations stereotype, have been provided in the class by means of the Opaque Behavior feature. Indeed, Close() and Release() operations have been used to describe of the Microwave oven door, modeling its behavior when it is left open or closed.

![Figure 5.10 - Microwave Oven's Door xtUML State Machine](image)

The ALF representation of each entry-behavior of the Door’s State Machine diagram (see Figure 5.10) is discussed below. The entry behavior body has been defined to any activity related to the appropriate state as ALF Unit [21].

As already stated, following ALF code is only a proposal since only the beta version of the ALF standard has been released.

5.5.1 Activities

According to the ALF Specification [21], every activity represents an activity node of the State Machine model.
/** Entry action for state 1: the microwave door is opened */
activity Open()
{
    this.isSecure = false;
    oven = this.'is accessed via';
    oven.CancelCooking();
}

The first assignment statement sets the Door class property isSecure to false, indicating the door lock-status when it is opened. The second and third statements, similarly to the select one statement of the OAL action language, select the unique instance of the oven along the Door-Oven relationship association (R4) and call the CancelCooking() method.

Different to 3G languages, such as Java/C++, the ALF language allows to use uncommon variables format, like the 'is accessed via' variable, enabling association references during action's specification. The R4 association concept, which has been defined through ALF in the system package, is showed later on this section. Performing these task an assumption must be made: the Oven instance need to be already created in one of the Oven class activities by using the statement new Oven(), where new Oven() is an instance creation expression semantically different to standard constructors [21].

Similar to the Open activity, the single statement provided in the Closed activity sets the property isSecure to true.

/** Entry action for state 2: the microwave door is closed */
activity Closed()
{
    this.isSecure = true;
}

5.5.2 The Microwave Model system package

ALF includes notation for structural models, such as packages, classes, and association, at the extended conformance level [21]. For this reason, the structural model of the Microwave Oven example can also be represented by using the ALF action language.

The system package and all imported external packages (referred by external-entities elements) defined into the structural model, have been represented by ALF in the same way they are defined in any Java project, that is by means of public classes. Therefore, the Microwave Oven system might be described as follows:

/** Microwave Oven */
package MicrowaveOven
public active class Door;
public active class Oven;

public assoc R4
{
    public 'is accessed via': Door [1..1];
    public 'provides access to': Oven [1..1];
}

However, differently from the Java package, in the ALF system package all associations of the structural model have to be specified, by using the assoc concept.

The ALF language also represents every class of the structural model by using the active class concept. Hence, the Door class representation might be described as follows:

/** Door class: active class to manage microwave's door */
active class Door
{
    public DoorID: unique_id;
    public isSecure: boolean;

    public receive signal Close();
    public receive signal Release();

    private activity Open;
    private activity Closed;
}
do
{
    accept(MO_D2: close);
    Closed();
    accept(MO_D1: Release);
    Open();
}

The first two statements represent the attributes specified in the Microwave Oven structural model by means of the proposed profile. The following two statements, declare the signals (events) defined in a generic State Machine model used by the private, Closed() and Opened() activities (active nodes). At the end, the body of the do statement, describes the activity flow by means of the accept statements and the call to the defined activity, here defined like procedures.
5.6 Case study evaluation

As presented in this section, the proposed profile and the ALF action language can be integrated each other to define UML system models, such as the Microwave Oven system.

A bidirectional mapping from the profiled to the xtUML models is then successfully enabled, as well as to the RSA models.

In the former case, all concepts defined through ad hoc tagged-values and constraints by the pivot language, are semantically equivalent to the respective xtUML modeling elements. Furthermore, the OAL action language used during behavior specification by xtUML models is also fully mapped to the ALF concrete syntax, providing complete translation of statements of the both action languages.

In the latter case, the mapping process might cause loss of information for those concepts specified in the RSA models that are not completely supported by fUML, such as association's multiplicity and datatypes operations. Hence, the tool chaining process between RSA and Papyrus might be partially compromised and extra information during model transformation, might be required. However, the presented case study has shown how same elements from different tools can be weaved by the profiled pivot language, satisfying execution model's standard requirements for simulation and testing aims.

Currently, is not so clear whether the use of ALF in combination with fUML can entirely fulfill these important tasks. The ALF action language is structurally well-defined and model-to-model (M2M) and model-to-text (M2T) transformations will be easily accomplished during tool chaining once the first, official version of the language is finally released. In fact, some modeling limits discussed in Sections 4, such as Interface definition and Primitive Types restrictions, may be actually by-passed using several stratagems, respecting both the xtUML and fUML specification. However, some others limits have been left pending and extra effort might be required by developers to finally provide a complete transformation process.
6 Conclusions

The purpose of this thesis work was to develop a pivot profile able to perform meta-model transformations from BridgePoint models to RSA and vice versa, enabling tool chaining. The work provides an ideal solution on how to achieve this task by using new, OMG standards together with the proposed profile. Most of those standards are still ongoing and the Papyrus project community planned to integrate them in the project shortly. Unfortunately, a complete definition and testing process might not be performed for the same reasons.

However, the developed Papyrus plug-in encapsulates both the pivot profile and customized features, providing a ready-to-use extension once the tool will be conform to the fUML and ALF standards. Therefore, the tool chaining process would be fully accomplished if UML Class Diagrams will be fully integrated to the fUML Action Language editor and Virtual Machine, providing model execution too those models defined by using the proposed profile. Indeed, its stereotypes correctly represent important concepts, such as classes, attributes, and relationships, successfully applied during the tool chaining process between the BridgePoint and Rational Software Architect modeling tool, satisfying horizontal transformation at meta-model level.

However, some not essential concepts might have been left out of the profile. Furthermore, as highlighted several times in this report, fUML and ALF are still two ongoing standards that might play an important role during model execution and tool chaining game. In fact, not all concepts have been already defined and improvements might radically change part of the current version. For instance, the Base UML (bUML) subset of fUML, defined to write its execution model, is not completely mapped to Java concepts, compromising some aspects of model execution.

Nevertheless, further developments for this work would be addressed on the profile improvement, either extending or adding structural concepts, according to the involved standards. In addition, once both the fUML and ALF standards will be integrated in the Papyrus project, model-to-code (M2C) transformation for model execution may also be provided by using transformation languages, such as Acceleo [10]. In this case, the entire system may be then completely tested and the model, finally executed.
Annex A - Customize Papyrus: the palette toolbar

As presented in Section 3.3, Papyrus [18] is a general-purpose tool that strongly supports customization activities. *Popup Menus* or *Palette Toolbars* may be added to its workbench, providing an easy to use environment where extra concepts are defined by the tool extensions (plug-ins). In the latest Papyrus release (July 2010), available in the Eclipse – Helios distribution, the graphical extension feature has been strongly improved from previous versions, where tricky ways used to be followed to accomplish this task and, sometimes, hand-made code had also to be specified. Developers can then provide all necessary information to customize the palette toolbar, where only the necessary profile-based gears can be defined.

According to the concepts described in Section 4 for the proposed pivot profile, a brief description on the customized palette menu is shown in Table A.1.

<table>
<thead>
<tr>
<th>Palette entry name</th>
<th>Referred Stereotypes</th>
<th>From UML Meta-class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConsRelationship</td>
<td>bpConsRelationship</td>
<td>&lt;&lt;Association&gt;&gt;</td>
<td>Create a new constrained relationship</td>
</tr>
<tr>
<td>Class</td>
<td>bpClass</td>
<td>&lt;&lt;Class&gt;&gt;</td>
<td>Create a new class</td>
</tr>
<tr>
<td>Datatype</td>
<td>bpDatatype</td>
<td>&lt;&lt;Datatype&gt;&gt;</td>
<td>Create a new user-data type</td>
</tr>
<tr>
<td>GeneralizationSet</td>
<td>bpGeneralization</td>
<td>&lt;&lt;GeneralizationSet&gt;&gt;</td>
<td>Create a new generalized set</td>
</tr>
<tr>
<td>Operation</td>
<td>bpOperation</td>
<td>&lt;&lt;Operation&gt;&gt;</td>
<td>Create a new bpClass's operation</td>
</tr>
<tr>
<td>ExtEntity</td>
<td>bpExtEntity</td>
<td>&lt;&lt;Class&gt;&gt;</td>
<td>Create a new external entity class</td>
</tr>
<tr>
<td>ReflexAssociation</td>
<td>bpReflexAssoc</td>
<td>&lt;&lt;Association&gt;&gt;</td>
<td>Create a new reflexive association</td>
</tr>
<tr>
<td>Association</td>
<td>bpRelationship</td>
<td>&lt;&lt;Association&gt;&gt;</td>
<td>Create a new association</td>
</tr>
<tr>
<td>Referential Attribute</td>
<td>bpRefAttribute</td>
<td>&lt;&lt;Property&gt;&gt;</td>
<td>Create a new referential attribute</td>
</tr>
<tr>
<td>AlternativeKey</td>
<td>bpAlternativeKey</td>
<td>&lt;&lt;Property&gt;&gt;</td>
<td>Create a new alternative key</td>
</tr>
<tr>
<td>PrimaryKey</td>
<td>bpPrimaryKey</td>
<td>&lt;&lt;Property&gt;&gt;</td>
<td>Create a new primary key</td>
</tr>
<tr>
<td>FullPropertyString</td>
<td>bpRefAttribute, bpPrimaryKey, bpAlternativeKey</td>
<td>&lt;&lt;Property&gt;&gt;</td>
<td>Create a new property string with primary and alternative key and references to attributes in other bpClasses</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------</td>
<td>-------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DomainPkg</td>
<td>bpDomainPKG</td>
<td>&lt;&lt;Package&gt;&gt;</td>
<td>Create a new package were only one ExtEntityPkg, DatatypePkg and SubsystemPkg must be specified inside</td>
</tr>
<tr>
<td>ExtEntityPkg</td>
<td>bpExtEntityPKG</td>
<td>&lt;&lt;Package&gt;&gt;</td>
<td>Create a new package were only external entity must be specified inside.</td>
</tr>
<tr>
<td>DatatypePkg</td>
<td>bpDatatypePKG</td>
<td>&lt;&lt;Package&gt;&gt;</td>
<td>Create a new package where only user-data type must be specified inside</td>
</tr>
<tr>
<td>SubsystemPackage</td>
<td>bpSubsystemPKG</td>
<td>&lt;&lt;Package&gt;&gt;</td>
<td>Create a new package where only user-data type must be specified inside</td>
</tr>
</tbody>
</table>

*Table A.2 – bpProfile palette menu elements*

When a new Papyrus model is created and the proposed profile applied to it through Papyrus plug-ins, the customized palette menu is provided to developers in the workbench.

Figure 5.1, presented in Section 5, shows the customized palette menu on the right side of the workbench.


