An Open-Source Pivot Language for Proprietary Tool-chaining

Stefano Cucchiella, Antonio Cicchetti, Federico Ciccozzi
School of Innovation, Design and Engineering
Mälardalen University, MRTC
Västerås, Sweden
stefano.cucchiella@gmail.com, [antonio.cicchetti,federico.ciccozzi]@mdh.se

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 deals 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 work reports the experience gained in industrial settings: problems arisen from the usage of different tools for different aims during the development process are discussed; moreover, a possible solution based on an open-source pivot language acting as a bridge to share information between different proprietary tools is illustrated.

I. INTRODUCTION

Modeling has always played an important role in every engineering discipline as a way to break down and reduce problem complexity, anticipate possible realization issues, simulate multiple solution choices avoiding the irreversible consequences of the real world. Similarly, in the software engineering and development field, the usage of formal, well-defined models of the system allows to radically break down project costs and developing time [4]. Not surprisingly, industry showed an increasing interest in Model-Driven Engineering (MDE) techniques as a way to implement efficient software development. In fact, MDE moves the efforts from a code-centric to a model-centric approach: relevant aspects of the system under study are represented through models, which in turn are exploited to cope with all the development tasks in an automated way, ranging from design to code generation to testing and maintenance.

From an industrial point of view, a complete development approach, from modeling to testing activities, can enable the achievement of goals stated in the requirements while meeting market deadlines [29]. However, possible lack of correctness into systems may strongly affect quality and performance of delivered products; for these reasons, industry usually exploits a constellation of different tools specifically developed to deal with particular tasks in a specific domain.

For instance, proprietary tools such as BridgePoint by MentorGraphic [22] or Rational Software Architect by IBM [14] adopt the Unified Modeling Language (UML) [27] as base language. Nonetheless, both of them specialize the UML by means of profiling mechanisms in order to give precise semantics to allow code generation as well as model execution. On one hand, such specializations improve tool performances enabling the generation of 100% of the implementation source code; on the other hand, tool-chaining and model interchange are hindered due to semantic gaps and/or mismatches. As a consequence, requirement models could be not exploitable during maintenance stages, and design models not reusable for testing purposes [7].

In this paper, we provide an overview on the main aspects of tools combination during the development phase together with arisen problems. Moreover, we propose a possible solution based on a pivot language acting as a bridge for the interchange of models across different proprietary tools. In particular, this work proposes to exploit a single source of information by which it is possible to provide appropriate inputs (or to get appropriate outputs) to (from) a selected tool through model transformations. Therefore, we do not rely on compatibility between tool formats, since it does not hold in general (as explained later on), but we rather build conceptual mappings between tools. Such mappings can range from simple element correspondences to semantic anchoring enabling information transfer to different applicative domains. It is worth noting that our methodology operates in well-defined boundaries, that is telecommunications domain. Therefore, it can be assumed that the pivot language will contain a coherent set of information and will not explode in its size; moreover, possible improvements will only require limited refinements of the existing mappings. In this respect, our proposal can improve the chaining resilience when tool updates occur, since correspondences are drawn at conceptual level and do not rely on (structured) textual format translations.

The remainder of the paper is structured as follows. Section II provides an overview on the main issues related to the use of several tools during the development process and presents three widespread proprietary tools exploited in the telecommunications domain. Moreover it introduces similar approaches present in the literature, for supporting our motivations and claims. According to the specification of those tools, Section III discusses the concepts behind the proposed pivot language, whose application to a selected case-study is unwound in
companies usually prefer them to the open-source ones, Most of the advanced modeling tools are proprietary and translated to other equivalent elements or not, especially in tool-chaining is to understand whether concepts can be independently from the adopted approach a common difficulty Rational Software Architect, is fully described. However, partially solve tool-chaining issues between BridgePoint and more platforms. In [8], an industrial example about how to prefer to use different tools for different development VI proposes some possible future works.

Section IV. Finally, Section V concludes the paper and Section VI proposes some possible future works.

II. BACKGROUND

The common process pursued by software developers is using different tools for different phases of the development, from analysis to testing. Having proper tools interoperability makes interchanging concepts semantically equivalent among the different tools and hence helps in avoiding modeling inconsistencies by guaranteeing a coherent development flow [11]. However, some important issues may arise during tools interweaving for interoperability aims [5].

A. Challenges in Tool-chaining

When dealing with tool-chaining, several problems arise; in this section we discuss the following problems that we identified as the most challenging:

- Data exchange among different tools
- Compatibility among different releases of a tool
- Proprietary nature of industrial tools

Data exchange. Since different tools most often use different data formats to represent information, one of the major problems achieving interoperability is strictly related to information exchange. XMI or XML are two examples of data formats used for such purpose [19] but, nowadays, their spread is affected by the new UML Diagram Interchange (UMLDI) [24] standard. In other words, bridge operations must be correctly defined in order to provide a bidirectional transformation among different formats since a standard has not been agreed yet.

Compatibility. Tool functionalities change rapidly and keeping trace of each modification in a software application is often not a trivial task. This can have a deep impact on those concepts that, defined by means of older versions of a tool, might not be fully representable in newer versions due to language core modifications, e.g. modifications to the meta-model. In this case, horizontal transformations and bridge operations at meta-modeling level must be able to keep trace of differences between the involved meta-models, providing a unique conversion [6]. Unfortunately such activity can be pursued only if the involved tool is not proprietary.

Proprietary nature. Since many industrial companies prefer to use different tools for different development phases, tool-chaining must be available between two or more platforms. In [8], an industrial example about how to partially solve tool-chaining issues between BridgePoint and Rational Software Architect, is fully described. However, independently from the adopted approach a common difficulty in tool-chaining is to understand whether concepts can be translated to other equivalent elements or not, especially when they are part of a proprietary tool.

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 tools due to proprietary and business restrictions. The interchanged model and its semantics can hence be partially invalidated, causing an incomplete translation of concepts between two different tools through model-to-model (M2M) transformations.

We hereby aim at providing an ideal solution for a pivot tool able to deal with these problems. Such solution has been intensively studied and currently important steps have been taken towards its complete specification. However, some features, such as model execution for testing and debugging purposes, are still under development and not far from completion.

Tools aiming at providing modeling of both structural and behavioral aspects are a small subset of the UML-compliance, modeling tools. Indeed, only few action languages exist to fully describe behavioral aspects [16] of a system model and most of them are proprietary and/or domain-specific. Examples of such languages are +CAL [28], mainly focused on distributed real-time systems, or Action Language for Business Logic (ABL) [12].

B. Selected Tools

For achieving our pivot language, three modeling tools widely used in the telecommunication domain have been selected as subjects: (1) Bridgepoint by MentorGraphics, (2) IBM Rational Software Architect v8.0, and (3) the Eclipse-project Papyrus.

BridgePoint is an advanced platform-independent, UML-based environment used by a large number of top companies for improving software modeling and testing. Thanks to its high-quality and code conformation to a large number of standards, a structural and behavioral formal model can be easily designed and an optimized domain-specific code generated. The structure of a generic system model is specified by means of the xtUML modeling language [21], while the behavioral aspects by using the Object Action Language [23], providing a unique environment for specification and testing aims. The development focus is thus shifted from coding, by using common 3G languages like C or Java, to higher level of abstraction. In fact, an abstract modeling language, such as xtUML, helps developers to decrease coding time by using model-to-code (M2C) transformations.

Similarly to BridgePoint, Rational Software Architect provides a complete environment for software development: structural and behavioral aspects are respectively described by using the UML standard and the UML Action Language (UAL) [2]. Model execution and simulation is one of the key features provided in this framework. Specifically, simulation can be performed either on informal or formal models, where UAL is used to describe behavioral aspects.

In any case, it largely shares features with BridgePoint,
providing automated development by using UML models and Domain Specific Modeling Languages (DSL) through UML profiling, in order to reduce the risk of failure. The framework also provides predefined M2M and M2C transformations, used to translate the specified system model to different targets, such as CORBA models or C++/Java code.

The tools described above are proprietary, hence costly maintenance, tool extension and customization are their main weaknesses. Papyrus has been recently released as an open-source UML2 modeling framework based on the Eclipse platform. It is an easily customizable general-purpose graphical editing tool, providing a complete support for UML profiling and model-based development. In fact, the main goal of Papyrus is to provide a valid alternative to proprietary tools, such as BridgePoint and Rational Software Architect.

Thanks to its capabilities, growing interest has been shown by both R&D sectors of industrial and academic research, which are working together to obtain a complete, Eclipse-based framework able to fully support design of secure and critical system [3].

The idea behind Papyrus is weaving statical and behavioral aspects into a canonical model, where the behavioral concepts are directly integrated into UML Class diagrams, which are the only ones that can be used to specify the system.

C. Related Work

In the literature several research attempts can be found in the direction of tool-chaining for model interchanging among numerous, semantically different tools. Nowadays, most of such approaches share a similar theory behind the interleaving of system static concepts, while concerns related to system behavioral aspects still represent the bone of contention among different research groups due to the lack of a uniform and standardized approach.

In [9] operational semantics is explicitly encapsulated into stereotype operations and semantic variation points are defined; the benefits derived from the usage of Foundational Subset for Executable UML Models (fUML) [26] strengthen the UML Profiling mechanism for the provision of a UML-compliant approach. In their previous works Lazar et al. [17], [18] proposed an fUML-based action language and its concrete syntax, referencing their work on [9]. However, their action language is runnable only on ComDevValCo, a well-specific framework developed for software component definition, validation, and composition. An approach similar to ours has been recently proposed in [10] where an implementation of fUML with Kermeta executable language is presented. However, just the definition of the fUML operational semantics has been successfully described without providing any concrete syntax for its elements.

Unfortunately, all the approaches listed above defined their own operational semantics and, when provided, the concrete textual semantics still remains unstandardized.

In [20] the authors present an automated framework for architectural languages and tools interoperability thanks to automated model transformation techniques. Such approaches in the literature show the urgent need of effective interoperability among different specific notations, tools, and domains in order to alleviate the developer’s effort in floating among different tools and representations for performing different actions in the development process.

Karsai et al. propose in [15] a pattern-based approach for tools integration that they try to address through a meta-model based approach. Meta-models for input and output models are defined and the semantic mapping between source and target tools or formats is represented by a model that is then used for the semantic translation. The approach can be applied either by integration of integrated semantic mapping models, which requires as many bi-directional translators as the number of tools to be integrated, or integration by process flow, which uses pair-wise tool integration. In the first case up to three tools can be integrated and all tools share a common model while, in the latter, the approach is able to integrate up to six tools and modifications pertain only to the two tools sharing the modified model [31]. In this respect our proposal does not rely on shared formats, since, as aforementioned, this is one of the relevant issues we faced in the industrial environment we deal with. On the contrary, models are aligned through the pivot language by means of model transformations, thus avoiding the need for compatible formats. Since the mapping is performed at conceptual level as opposed to the textual one, we believe that our solution can offer improved resilience to tool evolutions, that typically do not corrupt existing features.

The CASE Data Interchange Format (CDIF) described in [30] is defined as a set of EIA standardized meta-models for modeling of information systems and their exchange among different CASE tools. The so-called semantic meta-models have to be mapped to the specific vendors’ concepts by means of importer/exporter tools, which are not provided. Developed to become a standard mechanism to support import/export style data integration, CDIF has turned to be considered a mechanism for transferring data between different data repositories, while our goal is to provide a pivot language able to act as a semi-transparent bridge between different proprietary tools, with no needs for implementing import/export facilities tailored to each involved tool.

For such a tool-chaining approach that spans multiple portions of the design cycle, it is necessary to achieve integration from different domains, or domain subsets. Still being general-domain, the selected subset of tools is likely to cover a set of concepts focusing on the telecommunications domain; this constraint allows us to ensure a certain level of semantic coherency concerning the concepts defined in the pivot language. In other words, if updates have to be performed on the pivot language due to a new tool to be integrated in the chain, the semantic variation would be in most cases negligible in the sense that its implementation cost would still be less than the cost of the manual process it is intended to replace [15].

Moreover, the approach is conceived to require 2*n map-
pings between \( n \) tools and the pivot language, hence avoiding the need of a geometric \( n^*n \) mapping among all the tools that would lead to a \( O(n^2) \) problem (Figure 1). This is achieved by defining the pivot language as a universal interchange language, similarly to the idea proposed in [30], which acts as bridge between the tools to be integrated. Therefore each tool relates itself directly and only to the pivot and remains unaware of the other tools in the chain. The bidirectional integration tool-pivot is achieved by means of M2M transformations that, depending on the degree of similarity between source and target, can end up in the generation of different views of the same concepts or the translation toward different concepts through semantic anchoring. An example of successful semantic anchoring for tools interweaving is presented in [7] and focuses on Model Based Testing (MBT). It provides an automatic generation of efficient test procedures directly from system models in order to accelerate the software development process; the authors show how to exploit efforts in modeling activities by automatically deriving test models from system models.

III. THE PIVOT LANGUAGE

As presented in Section II, one of the main problems in tool-chaining is related to the specification of concepts since, even if represented in different ways in source and target models, they may carry the same semantic information. Extensions and restrictions of modeling elements are the mechanisms used to provide well-formed, complete bridging operations.

The evaluation of semantic and syntactic gap discussed in this section will help in understanding the challenges to be faced in a tool-chaining process.

A. Compliance to xtUML and fUML

Executable and Translatable UML (xtUML) is widely used to design models and enrich them with action semantics. Being a proprietary language, it does not allow common users to have complete control on the aspects defined in its specification, both structural and behavioral. Being not able to specify certain concepts by simply using available modeling elements, the developers must find other ways to define what is needed, i.e. definition from scratch on new modeling elements. In some cases, such new definitions may generate concepts which are not semantically equivalent to the intended ones.

Since fUML can be seen as a subset of UML specification, not all the UML concepts can be used to specify fUML-compliant models. In order to create a model simultaneously conforming to fUML and xtUML, particular attention must be paid to some of their concepts: for instance, since xtUML’s Association Class and Interface concepts can not be used in a fUML model, a different way to represent them must be provided. Hence, designers should be able to draw system elements only by using packages, classes, properties, associations, and operations. At the same time, the system behavior should be defined only through activities and their related concepts expressed by the ALF action language. In fact, as stated in Section II, the idea behind the Papyrus project, is to use a single canonical class model to describe both architectural and behavioral concepts, achieving tool-chaining among the described frameworks (Figure 2).

According to the Action Language for Foundational UML models (ALF) standard [25], that provides a unique methodology to describe system behavior, and the fUML standard, we propose a new idea for achieving tool-chaining through models interoperability by the definition of an open-source standards-compliant pivot language.

B. The structural view

The UML Class Diagram is commonly used to create the system model structure due to its high degree of usability. However, as already discussed, Bridgepoint uses rather xtUML to represent the static view of system models. Hence, choosing the UML Profiling modeling approach, additional information have to be defined into the pivot profile during tool-chaining, by means of stereotypes, tagged values and constraints in order to be able to describe all the needed concepts.

In the followings, the main concepts enveloped by the pivot profile are fully described.

As shown in the selected xtUML case study (Section IV), four additional primitive types and their related operations have been embedded into the pivot profile:
real represents all numbers with fractional part, i.e. 1.0 or 2.5 values, and several arithmetic, unary and comparisons operations have been provided, similarly to the Integer primitive type.

unique_id is used to represent unique instance values of system’s attributes. In this case, two comparison operators are available: $=, \neq$.

inst_ref<$\text{Mapping}$> refers to data structures defined by another xtUML model and passed by reference to the current model. Similarly to unique_id primitive type, two comparison operators are available for this primitive type: $=, \neq$.

inst<$\text{Mapping}$> represents the data structure passed to the current system by an external xtUML model. All the comparison operators can be applied to this primitive type: $=, >, <, \neq, \leq, \geq$.

Each xtUML Package represents a particular sub-domain of the system; such organization in several packages allows the developer to model each concept in its appropriate system domain. There are four different types of xtUML packages in which unique elements are used to build the system model. Every package contains additional information to the UML Package meta-class, depending on its aims. For example, the Domain, External-Entity and Datatype packages do not provide any other additional information. On the other hand, the Subsystem package contains additional information regarding its identifier and the minimum and maximum identification number used by the classes in it defined. Such identification mechanism is used to uniquely identify every system model element during model execution by using Bridgepoint. Even though this information is directly controlled by ALF through object instantiation statements, in our proposed profile its specification must still conform to the xtUML meta-model, since each package’s identifier participates in building up the class identifier.

Similarly to primitive types, user data-types can be added by the developer into the pivot profile specifying the referred primitive type, if needed.

The Class concept is crucial in our pivot profile, since only classes can be used to represent elements having different purposes, according to the Papyrus specification. A class is related to other classes by means of well-specified associations which may include both stereotyped and UML standard attributes, according to the xtUML specification.

Furthermore, additional tagged values have been added to the Class concept to specify its property string, an identifier value which is usually shown under its name bounded by curly brackets (i.e. $<$property_string$>$) into xtUML models. The property string carries two important pieces of information, i.e. its identifier number and its key_letter, which uniquely identify the class into the xtUML model and are very useful for model execution purposes, similarly to the Package concept.

All the necessary model elements defined as UML elements, or hand-written code by an external system, must be represented by using the External Entity concept. It extends the UML meta-class Class without providing any other information but its name. Usually, only operations are specified into the system model for external entities during the modeling phase. In our pivot profile these operations must be stereotyped using the Bridge stereotype, as described later in this section.
The **Attributes** stereotype carries more information than standard UML Properties. Additional information to identify both primary (1) and alternative keys (e.g. \{12, 13\}), is used as tagged values in the pivot profile through specific stereotypes.

Furthermore, attributes may be specialized as **Referential Attributes** 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 in Entity-Relationship models.

Consequently, all stereotyped attributes must be owned by stereotyped classes. This rule is enforced by the following OCL constraint specified at meta-modeling level: `self.base_Property.owner.getAppliedStereotype('Profile::bpClass') <> null`.

Any **Relationship** association in our pivot model is identified through an identification number, which uniquely pinpoints the association within the whole system model.

Since xtUML defines only four cardinality values, the association concept into our pivot profile must be able to provide limited information about its multiplicity: **one_one**, **one_many**, or **many_many** are the only applicable values during association definition with further information about the involved association side. For example, setting an association with **one_many** value and condition side **left**, represents the multiplicity 0..1:1..*.

Furthermore, associations can be specialized either as **Reflexive** or **Constrained**. The former is used to represent auto-referenced associations with instances from the same class, while the latter represents constrained associations where the correct instance is defined among the involved specified relationships.

According to the fUML standard, an important role is played by stereotyped relationships. Similarly to UML, xtUML needs **association classes** to define the association relationship between classes. However, common UML/xtUML association classes are not allowed in fUML; so, when an association relationship must be defined as Association Class, constrained associations and multiplicity have to be used instead, as shown in Figure 3.

Similarly to the stereotyped relationship, the **Generalization** stereotype also needs a unique identifier. Its value must be different both from other Generalization and standard associations.

Moreover, when a new generalization association is specified into the model, sub-classes should 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” [27]. Adherence to this rule is achieved by setting the common UML properties **disjoint** and **complete** to **true**, into the stereotyped concept.

Every **Operation** stereotyped element in xtUML can be specified as either **instance-based** or **class-based operation**. This categorization is applied at modeling level since the actual specification of the operation is delayed to the implementation phase, where the concrete syntax from the action language is defined.

The **Operation** stereotype concept is also generalized as **Bridge** operation, which is used to represent operations defined into external-entity classes. It has the same features as the basic UML operation element. In any case, operations and Bridge operations can be applied only to stereotyped classes and external entity respectively: such constraining rules are specified into the profile by means of natural language.

The **Parameter** stereotype provides additional information, i.e. passing mode (by-value or by-reference), enabling the action language to choose the right concrete syntax.

### C. Behavioral aspects

By using ALF, concrete syntax can be directly applied to the pivot profile to define the behavioral concepts needed during tool-chaining: a successful mapping of concepts can hence be achieved. Therefore, transformations from ALF abstract syntax meta-model to fUML abstract meta-model are also provided, migrating "root objects from the Alf abstract syntax representation to UML elements" [25]. Hence, ALF may be used both in State-Machine and Component diagrams integrating its concrete syntax text to the profile’s elements.

Nevertheless, during the mapping process from xtUML-OAL to UML-UAL concepts, some mismatching problems related to concepts definition, may arise. That is why we define a pivot language based on the combination fUML-ALF, which is able to fully describe those concepts obtained through semantic differences between their respective meta-models and therefore allows to specify a bidirectional transformation between xtUML-OAL and UML-UAL allowing model interchanging between Bridgepoint and Rational Software...
IV. A CASE STUDY: THE MICROWAVE OVEN

In this section the Microwave Oven (MWO) system, available in the BridgePoint UML Suite Help, is used to validate the proposed pivot profile. The static concepts of this system are described by means of xtUML and the behavioral aspects through State-Machines where action semantics is defined in terms of OAL. By using our profile and integrating it with the new ALF action language, a new model semantically equivalent to the original xtUML model is generated under the Papyrus platform.

Once the profile is successfully validated by Papyrus, it can be launched as an embedded plug-in in a new Papyrus instance. The profile is then available into the registered profiles list when a new Papyrus model is created, together with a customized palette that helps in creating profile modeling entities by simple drag-and-drop.

Looking at the MWO system specified in xtUML, six packages are needed to describe both static and behavioral aspects: (1) the Domain Package is used to organize the packages in the system model, (2) the Datatype Package is used to describe data types based on the primitive data type, (3) the Function Package organizes the functions in the model and related to model simulation in Bridgepoint, (4) the External-Entity Package, organizes models or pieces of hand-written code specified outside of the current model, (5) the Subsystem Package is used to organize classes of the static view and associations in the MWO system, and (6) the State Machine Package is used to organize the states, events, transitions and activities for an active class, a class that has a behavior.

However, according to the mapping discussed in the previous sections, only four packages (Domain, Datatypes, External-Entity and Subsystem) have been specified in the pivot profile to organize the needed modeling concepts (Figure 4).

A. Static view

Only one package in the MWO system model is stereotyped as Domain package, in which all others modeling stereotyped elements can be added to or refer to each other.

The Datatypes package contains datatype and enumeration elements. Three user-defined datatypes have been specified by using the bpDatatype stereotype, adding more information to the standard UML Datatype concept, such as the referred primitive type. Furthermore, an Enumeration element type has been specified by using the UML Enumeration concept and it is used to define the tube wattages levels in the MWO system.

External-Entities elements have been added to the model through the bpExtEntity stereotype. The MWO has three external entities, which describe model’s external functionalities, such as the control panel or the timing system, and which are not directly specified in the current model. Such entities are strictly related to testing activities performed into the BridgePoint environment. Since fUML and ALF are still on-going specifications a full validation on bridge operations can not be performed yet.

Subsystem package contains core elements of the system model, represented through a class model. Stereotyped classes, attributes and associations are related to each other and are maintained semantically conforming to xtUML. Classes are specialized by using the bpClass stereotype, which provides additional information about the reference number and the key letter used to identify each class. According to the MWO system specification, these two tagged-values are respectively set to 1 and MO_O. Each attribute requiring the property string are extended by means of the bpAttribute stereotype: in this case, isPrimaryKey is the only property set to true for the OvenID attribute. On the other hand, the referred attributes (TubeID, LightID, BeeperID, DoorID and TurntableID) are specified by bpRefAttribute stereotype, which includes the association to which the attribute is related by using the ref property, as shown by the DoorID attribute of the Oven class in Figure 5.

At last, each association in the class diagram is specialized by the bpRelationship stereotype, as for the R4 association in Figure 6. As stated in the previous section, not all the possible multiplicity combinations can be specified using the pivot profile. Therefore, the needed information and related constraining rules are inherited at modeling rather than at meta-modeling level.

This case-study is mainly focused on the Door class and how to represent and integrate its state machine using ALF. Figure 7, shows part of the MWO Class Diagram designed by
B. Behavioral view

The operation compartment of the Door class (Figure 7) contains two stereotyped operations which have been declared according to the case study’s description. Both functionalities represent the node elements used into the xtUML State-Machine model of the MWO system and provide behavioral functionalities to the stereotyped class, such as opening and closing actions.

The related OAL syntax has been then replaced by the ALF concrete syntax, defined as OpaqueBehavior in the operation body of the active Door class. In fact, according to the ALF Specification, each activity represents an activity node of the State-Machine model.

Thus, the related OAL code of the Open operation, specified into the State Machine model as:

```plaintext
event{
    assign_self.is_secure = false;
    select one oven related by self->MO_O[R4];
    generate MO_04: ‘cancel cooking’ to oven;
}
```

is replaced by the following ALF concrete syntax:

```plaintext
activity Open()
{
    this.isSecure = false;
    oven = this.‘is accessed via’;
    oven.CancelCooking();
}
```

The first assignment statement sets the class-model property isSecure to false, indicating the door safety status when it is opened. The second and third statements, similarly to the select one statement in the xtUML Microwave Oven model, select the oven instance from the Door<->Oven relationship (association R4) and call the CancelCooking() method.

In particular, the ALF language allows to use uncommon variables, like is accessed via, which indicates how the referred class is accessed by means of its relationship label rather than its name. For instance, the R4 relationship is specified as into the ALF System Package, together with the definition of other classes.

```plaintext
package MicrowaveOven
{
    public active class Door;
    public active class Oven;
    {
        public ‘is accessed via’: Door [1..1];
        public ‘provides access to’: Oven [1..1];
    }
}
```

Note that, in order to perform these tasks, an assumption must be made: the oven object must already be created in one of the Oven class activities by using the statement:

```plaintext
oven = new Oven();
```

where new Customer() is an instance creation expression, semantically different from a standard constructor.

Closed activity assigns and sets every needed variable to provide the closing functionality for the oven’s door, similarly to the Open.

The ALF System Package contains all the imported external packages referred by the external entities elements defined in the structural model, and are represented by ALF similarly to standard packages representation in Java. Differently from Java packages, all the associations of the structural model must be specified, by using the assoc concept. Finally, each class of the structural model must also be represented by the ALF language, by using the active class concept. The ALF class representation of Door class can be described as follows:

```plaintext
active class Door
{
    public DoorID: unique_id;
    public isSecure: boolean;

    public receive signal close();
    public receive signal release();

    private activity Open;
    private activity Closed;
}
```


The first two statements represent the attributes specified in the MWO structural model by means of the proposed profile. Then, two statements declare the signals (events) defined in a generic State-Machine model, for the two private activities (active nodes) defined afterwards. Eventually, the body of the accept statement describes the activity flow by means of the accept statements and the call to the defined activity through the inherit signal, defined here as procedures.

V. Conclusion

In this paper we proposed our experience in achieving a solution for tackling the issues arising from proprietary tool-chaining in industrial settings and showed its capabilities through a toy example. Moreover, there exists an ongoing validation process against industrial models which can not be shown due to property preservation; nonetheless, as also demonstrated by the MWO case-study, the proposed pivot profile appears as a promising start towards the definition of a bridge-language for effective interoperability among different notations and tools, aiming at helping the developer in freely surfing among different tools and representations during the software development process. Furthermore, since fUML is going to be part of the Papyrus project in the near future, the fUML-compliant ALF action language may result much useful in supporting its capabilities.

The pivot profile defined in this paper should be seen as a starting point towards a complete solution to the problems encountered in the tool-chaining process. The concepts defined in it have been carefully specified according to the fUML specification and by means of a Papyrus-plugin, in order to make it ready-to-use, as soon as all the involved standards will be delivered.

The idea behind the proposed profile is integrating the concept of UML Class Diagram and the capabilities given by the fUML's Action Language and Virtual Machine; such integration allows the execution of the models defined by means of the profile itself. The defined stereotypes represent the core modeling concepts, while other additional information for classes, attributes, relationships, datatypes have been defined as tagged values; eventually, an application of the profile has been shown through the MWO case-study.

VI. Future Works

As stated along within the paper, fUML and ALF are still ongoing standards that may play an important role in both sides of model execution and tool-chaining. Not all their concepts have been defined yet and improvements might radically change part of the current versions. For instance, the Base UML (bUML) subset, defined into the fUML specification in order to bound its execution model, is not completely mapped to Java concepts and therefore some capabilities of model execution features might be jeopardized.

Further enhancements of the proposed pivot language include improvements of the profile's concepts, either extending or adding structural concepts according to the involved standards. In addition, once both fUML and ALF standards will be integrated in the Papyrus project, M2C transformations for model execution could also be provided by using appropriate model transformation languages, such as Acceleo [1]. At that point, the entire system could be completely tested and the models executed. Moreover, a possible integration with the approach proposed in [7] where a syntactic and semantic transformation from xUML to Qtronic models [13] is under evaluation. Such integration would extend the context embraced by our pivot profile towards MBT concerns by allowing a direct transformation from pivot-profiled models to Qtronic models.

ACKNOWLEDGMENTS

The research work from which this paper has originated was partially supported by the CHESS project (ARTEMIS JU grant nr. 216682) [3]. The authors would like to thank Pär Asplund for his support and fruitful discussions on the definition of the proposed solution.

REFERENCES