A UML-Based Domain Specific Modeling Language for the Availability Management Framework

P. Salehi, A. Hamou-Lhadj, P. Colombo, F. Khendek
Electrical and Computer Engineering Department
Concordia University
Montréal, Canada
{pe_saleh,abdelw,colombo,khendek}@ece.concordia.ca

M. Toeroe
Ericsson Inc.
Montréal, Canada
Maria.Toeroe@ericsson.com

Abstract— **The Service Availability Forum (SA Forum) is a consortium of several telecommunications and computing companies that defines standard solutions for high availability platforms. One of the most important SA Forum services is the Availability Management Framework (AMF) which is responsible for managing the availability of an application running under its control. To achieve this, AMF requires a complete configuration, which consists of several entities organized according to AMF rules and constraints. In this paper, we argue that AMF concepts form a domain for which a domain-specific modeling language can greatly facilitate the generation, analysis and the management of AMF configurations. We define such a language by extending UML through its profiling mechanism and we implement it. More important, we discuss the challenges and the lessons learned in the course of this project.**

Keywords- High availability; Availability Management Framework, Configurations; Domain-specific modeling languages; UML profiles.

I. Introduction

The growing reliance on computing platforms has led to an increase in the customer's demand for robust and safe systems. For such systems, the requirement of providing services with minimal to no interruptions has become essential. The development of highly available systems and applications has been investigated for several years and several solutions have been proposed (e.g. [1]). However these are proprietary solutions, and thus hinder the portability of the applications.

The Service Availability Forum (SA Forum) [2] is a consortium of telecommunications and computing companies working together to define and standardize high availability solutions. SA Forum has developed the Application Interface Specification (AIS) [3] to support the development of Highly Available (HA) applications. The Availability Management Framework (AMF) [4] is among the services defined in AIS. AMF is responsible of managing the availability of the services provided by an application. It manages the redundant components of an application and can dynamically shift the workload from the faulty components to the healthy ones. AMF requires a complete configuration of the application. This configuration consists of a set of logical entities organized according to rules and constraints defined in the AMF specification [4]. AMF concepts form a domain for which a domain-specific modeling language (DSML) [5] can greatly facilitate the generation, analysis, and the management of the configurations.

With a new UML-based DSML for AMF, one can benefit from the advantages of a domain specific modeling language (e.g. usability, reuse & conservation of domain knowledge and ease of communication) as well as from the advantages associated with UML (e.g., standard design notation, tools support). However, UML is a general purpose language; it is far too general to capture the concepts in the AMF specification directly. Therefore, we decided to extend UML by defining UACL (UML-based AMF Configuration Language), a UML profile for the modeling of AMF configurations. This profile captures the complete and comprehensive definitions of all the domain concepts, their attributes, their relationships and the domain specific constraints. UACL has been implemented using IBM RSA [6] and represents an important contribution to the service high availability community since it aims at enabling different activities, such as the automatic generation of valid AMF configurations, the validation of third party AMF configurations and the analysis of nonfunctional characteristics such as availability.

Throughout the development of the profile, we faced several challenges due to a lack of a systematic approach for creating profiles, especially for the mapping of the domain concepts to UML meta-classes. In many situations, we have found that there were many alternatives from which it was not always obvious which one to choose. In addition, the use of Object Constraint Language (OCL) [7] turned out to be problematic, since most of the domain constraints needed extensive OCL expressions that cross-cut several domain contexts. Moreover, tools did not help either, and several tools that we have tried did not provide sufficient guidance. They also lacked ways to effectively validate the constraints. These challenges are discussed along with the lessons learned from the overall project, with the aim of contributing to the modeling community with the results of this experience.

The remaining part of this paper is structured as follows. In Section 2, we briefly introduce the main concepts in the

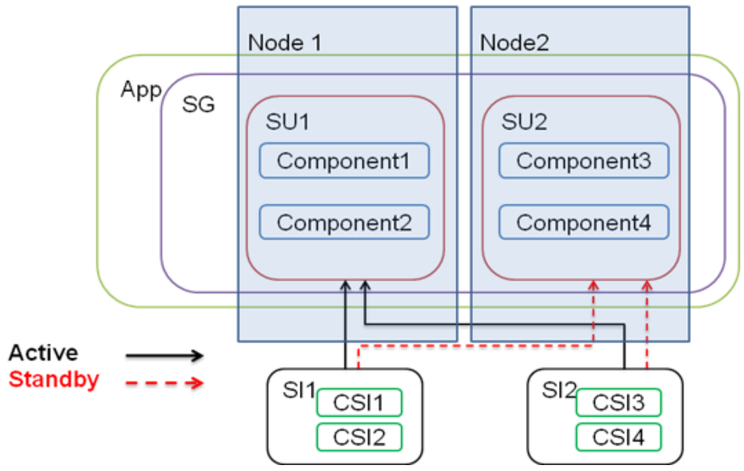

Figure 1. An example of AMF configuration

AMF specification. We describe the methodology we have used for the design of our profile, the domain model of the profile as well as the description of the language and its mapping to UML meta-model in Section 3. In Section 4 we present the lessons learned and the applications of UACL, followed by a review of related work in Section 5. We conclude the paper in Section 6.

II. THE AMF CONFIGURATION DOMAIN

AMF [4] is part of the AIS middleware, responsible for managing the availability of the services provided by an application. AMF fulfills this responsibility by managing the redundant components of an application by dynamically shifting workloads of faulty components to the healthy components. An AMF configuration for a given application is a logical organization of resources that enables AMF to perform workload assignments to provide service availability [3]. An AMF configuration consists of two different set of elements: AMF entities and AMF entity types.

The basic entity of an AMF configuration is the AMF component, which represents a set of software/hardware resources that can provide basic services. The workload assigned to components is represented as component service instances (CSIs). In order to combine the functionality of several components into higher level services, the components are logically grouped into service units (SU). Similarly, components service instances are also aggregated into higher level services that are referred to as service instances (SIs). SUs are aggregated into logical entities called service groups (SGs) in order to protect their services. Using redundancy models, an SG protects a set of SIs that is assigned (active or standby) to its SUs. When a particular SI is assigned to an SU, its composing CSIs are assigned to the components of this SU. An AMF application is the combination of service groups. From a deployment perspective, each SU is deployed on an AMF node and the set of all AMF nodes forms the AMF cluster. However, nodes can be grouped into smaller sets called node groups which can act as host for SGs.

Fig. 1 shows an example of an AMF configuration. In this example, a cluster is composed of two nodes (Node1 and Node2). It hosts an application consisting of one service group protecting two service instances in a 2N redundancy model. The service group consists of two service units, SU1 and SU2, each composed of two components. The distribution of the active and standby assignments is shown in this figure. This assignment happens at run-time and not at configuration time.

AMF entity types represent limitations and constraints imposed on AMF entities. There is type object for each AMF entity object except AMF clusters and nodes. The types are derived from a vendor's description of the application, which is provided in the form of an Entity Type File[8].

III. BUILDING THE PROFILE

The definition of UACL is composed of two phases. The first phase is concerned with specifying the domain model of the profile, which formally describes the concepts of the AMF domain, the relationships among them, as well as the domain specific constraints. The second step consists of mapping the AMF domain model to the UML meta-model by defining a set of stereotypes, tagged values and constraints. This phase requires identifying the most appropriate UML concepts, represented as UML meta-classes, which needs to be extended to support the AMF domain concepts. The defined extension must : 1) be complete by containing all the elements needed by the domain; 2) not contradict or violate the UML meta-model; 3) reuse meta-classes based on their semantics; 4) reuse as many UML relationships between the stereotyped elements as possible; 5) constrain the stereotyped elements to behave according to the rules of the domain.

A. *Defining the AMF domain model*

The AMF domain model has been developed by studying the AMF specifications and through constant interactions with a domain expert. The AMF domain elements are modeled as UML Classes. In addition, the relationships among them are modeled through different types of UML relationships. The constraints on the AMF domain model elements have been specified using the OCL [7].

As discussed in the previous sections, AMF concepts are classified into AMF entities and AMF entity types. Accordingly, we group such concepts into two packages named AMF Entity and AMF Entity Type. A further classification distinguishes the entities that provide the services (included in the Service Provider packages) from the services themselves (in the Service package). Similarly, two packages called Service Provider Type and Service Type have been defined to capture the AMF entity types. In addition, the AMF Entity package includes the Deployment package, which contains elements corresponding to the cluster and the nodes. There is no corresponding type package for the deployment package since the deployment entities are not typed. The following sections present the key AMF model elements which have guided the design of the UML extension for AMF.

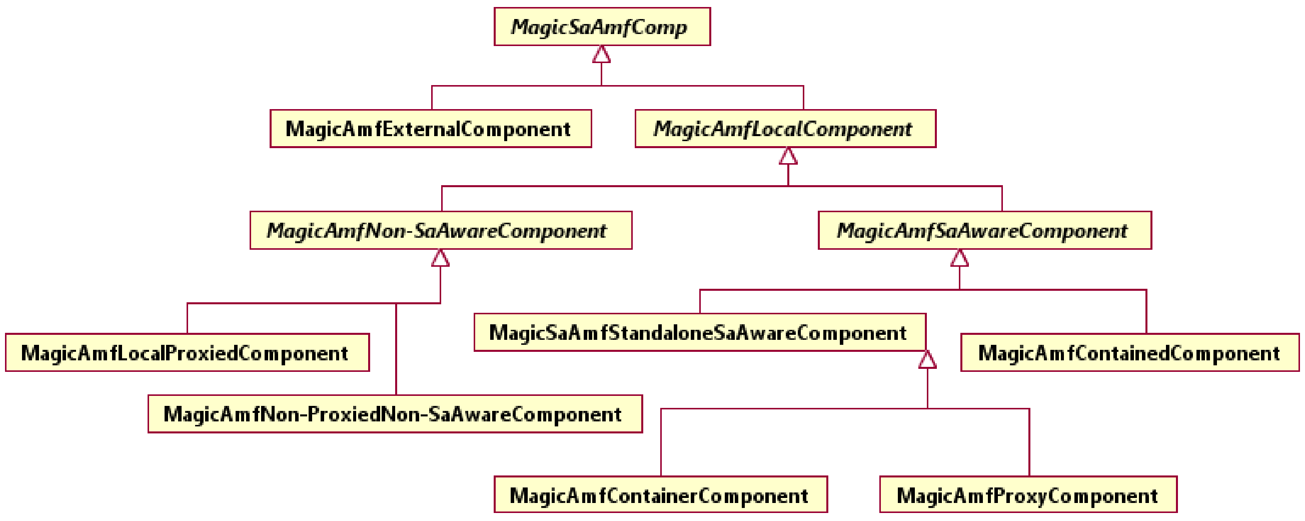
MagicSaAmfComp
MagicAmfExternalComponent
MagicAmfLocalComponent
MagicAmfNon-SaAwareComponent
MagicAmfSaAwareComponent
MagicAmfLocalProxiedComponent
MagicSaAmfStandaloneSaAwareComponent
MagicAmfContainedComponent
MagicAmfNon-ProxiedNon-SaAwareComponent
MagicAmfContainerComponent
MagicAmfProxyComponent

Figure 2. AMF Component Categories

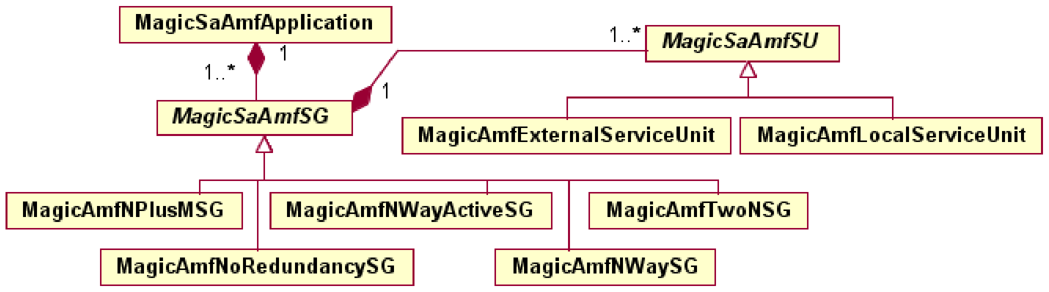

Figure 3. Service Unit and Service Group Categories

a) *AMF Components and Component Types*

Although AMF implicitly defined several categories of components, they are represented in the AMF specification as one aggregate element. We decided to classify AMF components according to four key orthogonal criteria: locality, service availability awareness (SA-awareness for short), containment, mediation, and (see Fig. 2). The SA-awareness criterion distinguishes the components that implement the AMF APIs and directly interact with an AMF implementation to manage service availability. SA-aware components are further specialized using other criteria. The containment criterion identifies the contained components that do not run directly on an operating system but use an intermediate environment, referred to as container component, like a virtual machine (for example, to support Java-like programs). Moreover, by using the mediation criterion, the SA-aware components are also classified into proxy and container components. Proxies are used to give AMF control over hardware or legacy software, called proxied components. Container components allow AMF controlling the life-cycle of contained components. Finally, the locality criterion distinguishes components that reside within an AMF cluster from the external ones. External components are also proxied to be controlled by AMF. The majority of components managed by AMF are expected to reside within the AMF cluster. The SA-aware components, regardless of the other criteria (containment and proxy-based mediation), are necessarily local. The local components category also includes the non SA-aware components which are either proxied or not proxied.

Unlike the component classification, our classification of the component types does not take into consideration the locality criterion. This is because the component type cannot specify whether its components have to be outside or inside of the AMF cluster. The component type class models the types of the SA-aware, the proxied components, and the non-proxied-non-SA-aware components. The SA-aware component type is further specialized to model the type of standalone components whose life cycle is managed directly by the AMF. Moreover, a standalone component type is further specialized into a proxy component type and a container component type which are the types of the proxy and container component, respectively.

b) SU, SG, SI, CSI and their Types

To provide a higher level service, components are grouped into service units (SUs). We distinguish between local and external SUs (see Fig. 3) based on whether or not they contain local or external components. SUs are organized into service groups (SGs) to protect services using different redundancy models: 2N, N+M, N-Way, N-Way-Active and No-redundancy. SGs are specialized based on the redundancy models used to protect their SIs (see Fig. 3). The original SG configuration attributes depicted in the AMF specification have been re-organized according to their relevance to the newly introduced SG classes. At the type level, the AMF specification defines an attribute to distinguish between the local and the external SU types. In our domain model, we specialize the SUTypes into two classes: MagicAmfLocalSUType and MagicAmfExternalSUType. The SGType and ApplicationType are the same as in the AMF specification as there is no specific reason to specialize them. The component service instance (CSI) and service instance (SI) entities are captured in our domain model as shown in Fig. 4.

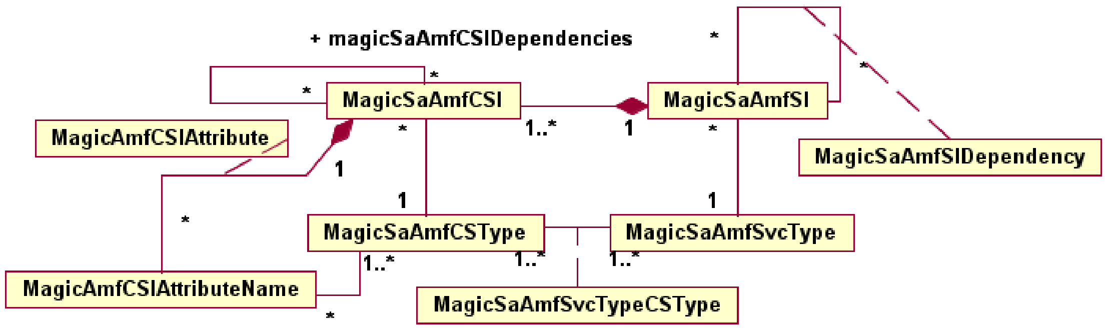

Figure 4. Component Service Instance and Service Instance

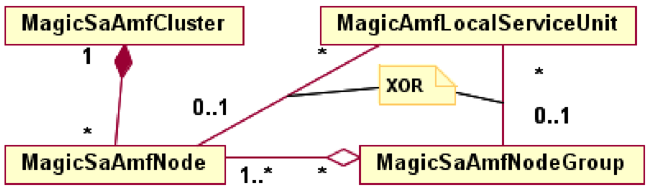

Figure 5. AMF Nodes, Node Groups, and Cluster

c) Deployment Entities

The cluster, the node and the node group represent part of our model for the deployment entities (see Fig. 5). An AMF cluster is a complete set of AMF nodes in the AMF configuration. A node represents a complete inventory of the SUs and, consequently, the corresponding components that it hosts. A node group represents a set of nodes and is used for the deployment of local SUs and SGs.

d) Domain Specific Constraints

We used OCL to describe the constraints on the AMF domain model elements. These constraints govern both the structure and the behavior of these entities. As an example of a constraint definition, let us consider the definition of the following property specified by the AMF specification: “the only valid redundancy model for the SGs whose SUs contain a container component is the N-Way-Active redundancy model”. This is expressed in OCL in the context of the container component category represented by the class MagicAmfContainerComponent, and by using our specific class for the SG associated with the N-Way-Active redundancy model, MagicAmfN-WayActiveSG. We can therefore easily capture this restriction in OCL as follows:

```
context MagicAmfContainerComponent
inv:
self.magicAmfLocalComponentMemberOf.
magicAmfLocalServiceUnitMemberOf.
   oclIsKindOf(MagicAmfN-WayActiveSG)
```

B. *Mapping the AMF Domain Model to the UML Meta-model*

Once the domain model is completed, the second step in designing a UML profile concerns the mapping of the domain concepts to the UML meta-model. For this purpose, one needs to proceed step-by-step through the full set of domain concepts, identifying the most appropriate UML meta-classes for each of them. Since UML 2.0 (the version used in this work) supports inheritance relationship between stereotypes, not all domain concepts need to be directly derived from the corresponding meta-classes. Instead, some of them can directly inherit from the newly created stereotypes.

1) Mapping AMF Domain Model Concepts to UML Meta-classes

This section presents the stereotypes for the previously defined AMF domain entities and entity types. For each stereotype a suitable meta-class is also presented.

a) Component

The component in AMF represents the encapsulation of the functionality of software that provides the services. This is similar to the concept of the component in UML, which is defined as “a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment” [9]. Therefore, we mapped the AMF component to a UML component defining a new stereotype called <<*MagicSaAmfComponent*>>. Similarly, a stereotype is defined for each component category and is indirectly mapped (through inheritance relationships between stereotypes) to the Component meta-class.

b) Service Unit

Based on the definition of SUs in the AMF domain, an SU is a logical entity that aggregates a set of components by combining their individual functionalities to provide a higher level service. From this perspective, one could see an SU as a service provider, similar to a component, but at higher level of abstraction. We therefore decided to map the SU to a UML Component meta-class as well. The stereotype <<*MagicSaAmfSU*>> is used to represent an SU. Local and external SUs are represented using the stereotypes <<*MagicAmfLocalServiceUnit*>> and <<*MagicAmfExternalServiceUnit*>>.

c) Service Group

One of the key characteristics of an SG is the grouping of SUs. At first glance UML package might seem to be a good meta-class candidate for SG. However, since an SG not only has the ability to contain other elements but also can offer a service, we decided to map it to the Component UML meta-class. The service offered by an SG can be seen as the ability to protect SIs that can be provided by its SUs. The stereotype <<*MagicSaAmfSG*>>, was created to represent an SG. A stereotype has also been created for each SG category, which derives from <<*MagicSaAmfSG*>>.

d) Application

An application is a logical entity that contains one or more service groups. An application combines the functionalities of the constituent service groups in order to provide a higher level service. Similar to a service unit, a UML Component has been found to be the most suitable base class for the stereotype designed to represent an AMF application (<<*MagicSaAmfApplication*>>).

e) Component Service Instance (CSI)

CSIs are representing attributes for services which are going to be provided by components. These attributes describe the characteristics of the workload which is going to be assigned to the component at run-time. In UML, “a class describes a set of objects that share the same specifications of features, constraints, and semantics”[9], and thus, the meta-class Class is semantically the closest meta-class to a CSI. As a result, it is used as the base class for the stereotype that has been defined for CSI (<<*MagicSaAmfCSI*>>).

f) Service Instance (SI)

An SI is an aggregation of all the CSIs to be assigned to the individual components of a service unit in order to provide a particular service. In fact, an SI shares most of the characteristics of the CSI but at a higher level of abstraction. Consequently, similar to CSI, the meta-class Class can be used as a base class for the stereotype defined for an SI (<<*MagicSaAmfSI*>>).

g) Node

A node in the AMF domain is a logical entity that represents a complete inventory of SUs and their components. We mapped the AMF node to the UML meta-class Node since, similar to AMF, a node in UML “is a computational resource upon which artifacts may be deployed for execution” [9]. We created the stereotype <<*MagicSaAmfNode*>> to refer to an AMF node.

h) Cluster and NodeGroup.

Based on the UML specification, “a package is used to group elements, and provides a namespace for the grouped elements” [9]. Moreover, the complete set of AMF nodes in the AMF configuration defines the AMF cluster. The role of an AMF cluster and node group is the grouping of different

AMF nodes. Therefore, the meta-class Package seems to be the most appropriate base class for the AMF cluster and node groups. The stereotypes *<<MagicSaAmfCluster>>* and *<<MagicSaAmfNodeGroup>>* are used to refer to these two entities.

i) AMF Entity Type elements

In general, the type of an entity describes the restrictions that should be respected by this entity. All entities of the same type share the attribute values defined in the entity type. Some of the attribute values may be overridden, and some other ones may be extended by the entity at configuration time. In other words, the type is the generalization of similar entities. For example, the service group type is a generalization of similar service groups that follow the same redundancy model, provide similar availability, and are composed of units of the same service unit types.

Considering the fact that, in UML, the meta-class Class describes a set of objects that share the same specifications of features, constraints, and semantics [9], it can be used as a base class for all AMF entity types.

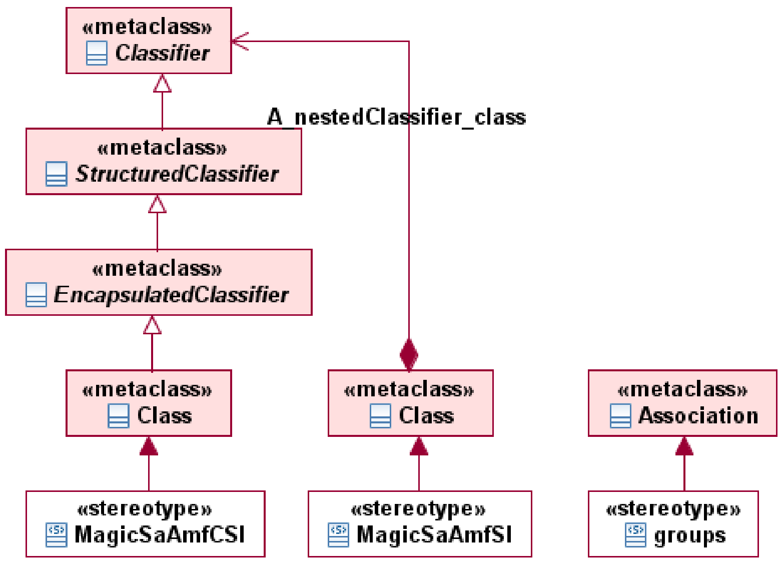

Figure 6. Relationship between SI and CSI

2) Mapping the AMF relationships to the UML Meta-model

In this section, we present the stereotypes that have been defined to capture the relationships among AMF entities, represented as UML stereotypes. We distinguish between six categories of relationships between domain concepts:

- *Provide*: This relationship is used between service providers and service elements and represents the capability to provide services.
- *Type*: It represents the relationship which is used between AMF entities and their type (e.g. the relationship between component and component type).
- *Group*: It represents the relationship which is used between grouping and grouped elements (e.g. the relationship between an SU and its enclosing components).
- *Protect*: It represents the relationship which is used between a service group and service instances in order to protect the services they represent. In other words, this represents the actual service provision.
- *Deploy*: It represents the relationship which is used for deployment purposes (e.g. between a service unit and a node or between service group and a node group).
- *Member node*: represents the relationship which is used between a node and a cluster or a node group

A careful selection of meta-classes for our domain concept related stereotypes allowed us to reuse many associations in the UML meta-model for the aforementioned relationships. Each association has been stereotyped accordingly and mapped to either Association, AssociationClass, or Dependency.

For example, both *<<MagicSaAmfSI>>* and *<<MagicSaAmfCSI>>* stereotypes are mapped to the UML meta-class Class and, since the meta-class Class inherits indirectly from the meta-class Classifier in the UML meta-model, there is an association between the classes Class and Classifier called "nestedClassifier" which allows classifiers to group other classifiers. We reused this association to express the fact that an SI (represented as *<<MagicSaAmfSI>>*) groups CSIs (represented as

<<*MagicSaAmfCSI*>>). Consequently, as shown in Fig. 6, we defined the stereotype <<*groups*>> to capture the relationship and map it to meta-class Association.

3) *Specifying Constraints*

This phase aims at ensuring that the UML stereotyped base meta-classes do not have attributes, associations, or constraints that conflict with the semantics of the domain model. If this is the case, UML itself needs to be restricted in order to match the domain related semantics and to guarantee the consistency of the profile with the semantics of the domain model. To this end, a set of constraints were defined. For example, the previously defined stereotype <<*groups*>> can be used only between specific AMF entities. However, UML has the capability of using association between all sorts of UML elements, including the meta-classes Class, Component, and Node. Therefore, without any constraints it would be possible to use the <<*groups*>> relationship to group component service instances into an AMF application, which is semantically invalid with respect to the AMF domain. Consequently, different constraints have been defined and expressed in OCL to restrict the UML meta-model in the context of AMF. For instance, the following constraint restricts the UML meta-model to use the <<*groups*>> stereotype between component and service unit:

```
context <<groups>>
 inv : (self.endType()->
        at(1).oclIsKindOf(MagicSaAmfComp)
or
        self.endType()->
        at(1).oclIsKindOf(MagicSaAmfSU))
and
(self.endType()->
        at(2).oclIsKindOf(MagicSaAmfComp)
or
        self.endType()->
        at(2).oclIsKindOf(MagicSaAmfSU))
and
        (self.endType()->
        at(1).oclIsKindOf(MagicSaAmfComp)
        implies
                self.endType()->
                at(2).oclIsKindOf(MagicSaAmfSU))
        and
                (self.endType()->
                at(2).oclIsKindOf(MagicSaAmfComp)
        implies
        self.endType()->
        at(1).oclIsKindOf(MagicSaAmfSU))
```

Another type of constraint is based on the AMF domain model: AMF components cannot inherit from other components, but the UML meta-model allows inheritance between elements that are mapped to the UML meta-class Component. Therefore, other constraints are required to restrict the standard UML elements to what is allowed by AMF. The following constraint restricts the inheritance on components.

```
context <<MagicSaAmfComponent>>
inv : self.general()->isEmpty()
```

IV. LESSONS LEARNED AND APPLICATIONS OF UACL

A. *Lessons learned*

After the analysis of the AMF domain and the design of the domain model, the first issue we faced was how to define the UACL language.

Although a UML profile may result in a less precise language than a MOFbased one, we avoided a MOF based solution as this suffers from a lack of tool support. The advantages of UML profile far outweigh its drawbacks [10]. The second issue was whether to extend existing profiles or to create a new one. We investigated existing profiles related to dependability and availability. We targeted both OMG standardized profiles, such as MARTE [11], SPT [12], and QoS&FT [13], as well as the non-standardized profiles reported in the literature, such as the DAM [14] profile and the profile presented by Szatmári et al. in [15]. The evaluation and analysis of these profiles were based on different criteria:

1. The capability of the profiles' constructs in capturing the concepts and semantics of the AMF domain and the complexity of extending these constructs when needed. The main goal of this extension is to take advantage of the profile features and reuse their constructs as much as possible. If the concepts of the AMF domain cannot be defined as a combination or extension of the basic constructs of the profile, the extension will be handled in the underlying UML meta-model. This outweighs the benefits of the extension. Most of the analyzed profiles turned out to be unsuitable. For example, the concept of service in the DAM profile addresses the description of the service itself, while in the AMF domain, the service is the description of attributes for the workload that will be assigned to service providers at run-time. In fact, there is a substantial distinction between the concept of service in DAM and in the AMF domain. Therefore, to capture this concept, we need to directly refer to the UML meta-model and not go through the DAM profile.
2. The implementation of the existing profiles. One of the goals of this work is to develop a CASE tool to support different activities, such as the design and validation of AMF configurations. In the case of extending existing profiles, we need to have access to their implementation such as the XMI format that serializes the profile model. We found that the non-standard profiles do not provide open access to their implementation. The implementation is available for some of the standard OMG profiles (for instance, for MARTE). However, due to the characteristics of the

AMF domain concepts, we could use only small fractions of these implementations. At the same time, building an extension requires importing and handling the whole implementation package. This may result in complexity at the tool development phase as well as performance issues at run-time. For instance, the run-time evaluation of newly defined constraints of the new language may require the evaluation of several constraints of the referred profile.

Because of the characteristics of the AMF domain and the fact that the required additional complexity does not justify the very few benefits of a possible extension, we decided to extend the UML meta-model instead of reusing another profile and adapting it to AMF. As a result, the most critical aspect of the design became the identification of UML meta-classes for mapping purposes. More precisely, we had to identify the most appropriate UML meta-classes to extend in order to support the AMF domain concepts. To the best of our knowledge, there is no systematic approach to guide this process. Selic [16] proposes the separation of the domain modeling phase and the mapping, but does not provide any guidelines for this mapping. Other studies [17, 18] propose patterns which are based on few types of relationships that may exist between domain elements and the corresponding meta-classes. However these guidelines focus on specific scenarios and do not provide a general solution to the mapping problem. In other words, no "ready to use" solution addresses the general issue of selecting the most appropriate UML meta-class for a specific domain element. Through the exercise of mapping AMF to UML, we have identified and used the following criteria that can guide an effective mapping process:

- *Semantic alignment*: given a domain concept, the selected meta-class has to capture the semantics of the domain concept. For our purpose, we have compared each AMF domain concept (specified in the domain model) with the UML elements in the UML superstructure specification. For each stereotype we found a set of possible options. For instance, consider the selection of the proper meta-class for a CSI. In the UML specification, a Classifier is an abstract meta-class which is a namespace whose members can include features. A BehavioralClassifier is a specific type of Classifier that may have an interface realization [9]. A behavioral classifier seems to be a good candidate for a CSI, since we can consider CSIs in terms of the workload which AMF dynamically assigns to components. However, it describes the characteristics of the workload which will be assigned to the component at run-time and not the service itself. Therefore, the meta-class BehavioralClassifier has been discarded. On the other hand, in UML, "a class describes a set of objects that share the same specifications of features, constraints, and semantics"[9], and thus, the meta-class Class is semantically closer to CSI.
- *Compliance*: This criterion aims at refining the choice of UML meta-classes that are semantically aligned with a particular domain concept. In fact, the newly defined stereotypes must not contradict nor violate the UML meta-model. For instance, based on the UML specification [9] "An *AssociationClass* cannot be defined between itself and something else". The <<*MagicAmfCSIAttributeName*>> in the AMF domain model, for instance, has an association to <<*MagicAmfCSIAttribute*>>. Therefore, we cannot map both stereotypes to the meta-class *AssociationClass*.
- *Reuse*: This criterion targets the refinement of the choice of possible meta-classes for each stereotype. The goal is to reuse as many UML relationships between the stereotyped elements as possible in the process of mapping to the UML meta-model. Reusing the associations among the meta-classes decreases the complexity of the design. Hence, if it is required to have a relationship between two stereotypes, it is better to reuse (if possible) the existing relationships between the corresponding meta-classes. For instance, in our profile <<*MagicSaAmfComp*>> and <<*MagicSaAmfSU*>> are mapped to the meta-class Component. According to the AMF domain, an SU aggregates a set of (AMF) Components. At the level of UML meta-model the reflexive association *packagedElement* is defined for the meta-class Component. Therefore, we have reused this association for the purpose of packaging between <<*MagicSaAmfSU*>> and <<*MagicSaAmfComp*>>
- *Constraints*: Constraints are defined to restrict the UML meta-model in order to capture the semantics of specific domain concepts. The selection must be performed with the goal of minimizing the number of constraints required at the level of the UML meta-model. This criterion is strictly related to the previous ones. For instance, mapping the <<*MagicSaAmfSU*>> to the meta-class Package (instead of Component) would require an additional constraint to restrict the merging feature provided by *Package*, since in AMF SUs cannot be merged.

The above mentioned criteria are interdependent and thus, cannot be handled separately. Moreover, the complexity in addressing them has a direct relationship with the size and the complexity of the domain. The development of UACL was complicated due to the large number of interrelated AMF domain concepts. The first two criteria were essential in designing UACL since they guided us through the selection of a semantically valid set of meta-classes. In other words, after applying the criteria we selected the meta-classes which were compliant with the AMF domain and aligned with the UML meta-model. Furthermore, the last two criteria helped us in mastering the complexity of the design as well as improving the quality of the profile by identifying the most appropriate mapping solutions.

In addition to the aforementioned design issues, a complementary and important aspect needs to be taken into consideration: The tool support. In [19], the authors compare different UML 2.0 integrated software development environments which support the design of UML profiles. This comparison is based on the capabilities of the tools such as integration with other tools and the effort required for defining a profile. We followed the conclusions of this paper and we based the implementation of UACL on IBM Rational Software Architect (RSA) [6]. The choice has not been based exclusively on the design capabilities; we also considered the facilities to support the different applications of UACL, such as the validation of AMF configurations. RSA provides this facility through the integration of an OCL interpretation engine. Moreover, through visualization and meta-model integration services, RSA can integrate different meta-models, allowing them to reference one another. Therefore, it enables the model-driven approach in which different meta-models can be involved simultaneously [6]. Consequently, it supports the usage of UACL as framework for the model based configuration generation, which is one of the applications we are currently targeting. However, our experience with RSA also showed some weaknesses when dealing with the implementation of OCL constraints. More specifically, to support the OCL functions that require access to stereotyped elements, RSA implements additional functions like *getAppliedSubstereotypes()* and *isStereotypeApplied()*. The main issue with these functions is that they cannot be interpreted using the live evaluation mode. Considering the fact that almost all of the constraints in UML profiles deal with stereotypes, this drawback has a great impact on the usability of the tool and on the performance. Indeed, the evaluation is performed exclusively in batch mode. Moreover, using tagged definitions in cross-context constraints is rather challenging. An example would be the specification of a typical OCL constraint in the context of one of the stereotypes associated with <<*MagicSaAmfSU*>> (e.g. <<*MagicSaAmfComp*>>) to restrict one of the attributes of <<*MagicSaAmfSU*>> –such as *magicSaAmfSURank*– not to have a value of zero. Despite its common occurrence, this constraint needs to be implemented using a complex expression such as:

```
self.ownedAttribute->
   select(ct:Property|ct.type.getAppliedSubstereotypes
   (MAGICAMFProfile::MagicSaAmfSU.oclAsType(
      uml::Stereotype))->notEmpty())->
      at(1).opposite.owner.oclAsType(uml::Class).
      getValue(MAGICAMFProfile::MagicSaAmfSU.
      oclAsType(uml::Stereotype),
      'magicSaAmfSURank').
      oclAsType(uml::Integer) <> 0
```

As presented above, accessing the attribute *magicSaAmfSURank* is only possible through a function called *getValue()*and through specifying the name of the stereotype and the tagged definition. In addition, at the end of the function we need to cast the type of the output of the function to *uml::Integer*.

B. *Application*

The implementation of UACL enables applications that will ease the development and analysis of AMF configurations:

- *Automatic AMF configuration generation*: UACL is being used for the development of a model based AMF configuration generation approach. The generated configurations are valid by construction. Moreover, such AMF configurations will be transformed into analysis models for the evaluation of their availability, and other non-functional characteristics. UACL will facilitate the transformation of configurations to analysis models.
- *Validation of AMF third-party configurations:* UACL can be used for the validation of AMF configurations developed by a third party. UACL can be seen as a formalization of the concepts, rules and constraints defined in the AMF specification against which a third party configuration has to be validated. This validation is simply performed through a successful or non-successful transformation of the third party AMF configuration to an instance of UACL.

V. Related Works

To our knowledge, the only study to extend UML to support AMF concepts is the one proposed in [15]. The authors' solution, however, suffers from many limitations: 1) in this work they deal only partially with the AMF domain, and the constraints on AMF model elements have not been included; 2) the presented UML extension process is only limited to specifying stereotypes, which allow adding new vocabulary to UML; they did not specify tagged values and constraints, which are critical to the extension mechanism; 3) their proposed profile is based on an older version of the AMF specification.

The literature reports only one other profile partially related to this domain, namely, the Dependability Analysis Modeling (DAM) profile [14] as an extension of MARTE to enhance its modeling facilities for analyzing dependability. In the DAM profile, the building blocks of a system are limited to components and services. Moreover, the concept of the service in the DAM profile addresses the description of the service itself while, in the AMF domain, the service is the description of the attributes for the workload which will be assigned to service providers at run-time.

VI. Conclusion

An AMF configuration is the artifact used by the AMF middleware for managing the high availability of services provided by applications under its control. In this paper we reported on the design of UACL, a UML profile for AMF Configurations and its implementation using the IBM RSA toolkit. The profile has been defined as an extension to the UML2 meta-model. The definition consisted of 1) the analysis of the AMF configuration domain, capturing all the

AMF domain concepts, 2) the definition of the concrete syntax of the language, and 3) the specification of the semantics through the mapping to the UML2 meta-model and the definition of constraints.

The experience, as discussed in Section 4, has shown that the most critical aspect of the design was the identification of meta-classes for mapping purposes. Due to the existing relationships at the level of the UML meta-classes, the selection of inappropriate base classes may result in the definition of a language that is not compliant with the UML semantics. UACL can support AMF configuration design, analysis and validation. Currently, it is being used with other models for the development of a model based configuration generation approach.

ACKNOWLEDGMENT

This work has been partially supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada and Ericsson Software Research.

REFERENCES

[1] Vogels, W. et al. The design and architecture of the Microsoft Cluster Service-A practical approach to high-availability and scalability. In: Int. Symp. on Fault-Tolerant Computing, IEEE (1998)

[2] Service Availability Forum, http://www.saforum.org

[3] Service Availability Forum, SA Forum Overview, SAI-Overview-B.05.01, http://www.saforum.org/specification/download.

[4] Service Availability Forum, Application Interface Specification, Availability Management Framework, SAI-AIS-AMF-B.05.01, http://www.saforum.org/specification/download/

[5] Kelly, S. and J. Tolvanen. Domain-specific modeling: enabling full code generation. Wiley-IEEE Computer Society Press (2008).

[6] IBM RSA, http://www-01.ibm.com/software/awdtools/architect/swarchitect/

[7] OMG, Object Constraint Language, Version 2.2 – http://www.omg.org/spec/OCL/2.2/PDF

[8] SA Forum, Application Interface Specification. Software Management Framework SAI-AIS-SMF-A.01.01, http://www.saforum.org/specification/download/.

[9] OMG, UML, Superstructure, v2.2., http://www.omg.org/cgi-bin/doc?formal/09-02-02.pdf

[10] Fuentes-Fernández, L. and A. Vallecillo-Moreno, An introduction to UML profiles. J. UML and Model Engineering, 2, (2004).

[11] OMG, UML Profile for MARTE, v1.0, http://www.omg.org/spec/MARTE/1.0/PDF

[12] OMG, UML Profile for Schedulability, Performance, and Time Specification, v1.1, http://www.omg.org/cgi-bin/doc?formal/2005-01-02

[13] OMG, UML Profile for Modelling Quality of Service and Fault Tolerance Characteristics and Mechanisms,v1.1, http://www.omg.org/spec/QFTP/1.1/

[14] Bernardi, S., Merseguer, J. and Petriu, D. Adding dependability analysis capabilities to MARTE Profile. In: MODELS 2008, LNCS v. 5301 (2008)

[15] Szatmári, Z., A. Kövi, and M. Reitenspiess. Applying MDA approach for the SA forum platform. In: 2nd Work. on Middleware-Application Interaction, ACM (2008).

[16] Selic, B. A systematic approach to domain-specific language design using UML. In: ISORC 2007, IEEE (2007).

[17] Lagarde, F., et al. Improving UML profile design practices by leveraging conceptual domain models. In: ASE 2007, ACM (2007).

[18] Lagarde, F., et al. Leveraging patterns on domain models to improve UML profile definition. In: FASE (2008), LNCS v. 4961 (2008).

[19] Amyot, D. H. and Roy, J. Evaluation of Development Tools for Domain-Specific Modeling Languages. In: 5th Int. Work. on System Analysis and Modeling, LNCS v. 4320 (2006).