A UML-Based Domain Specific Modeling Language for Service Availability Management: Design and Experience

P. SALEHI[1], A. HAMOU-LHADJ[2], M. TOEROE[3], F. KHENDEK[2]

[1]*Faculty of Applied Science and Technology*

Sheridan College

Oakville, Canada

[2]*Electrical and Computer Engineering Department*

Concordia University

Montréal, Canada

[3]*Ericsson Inc.*

Montréal, Canada

pejman.salehi@sheridancollege.ca

abdelw@ece.concordia.ca

maria.toeroe@ericsson.com

khendek@ece.concordia.ca

Abstract

For most critical systems, the requirement of providing services with minimal to no interruption is essential. The Service Availability Forum (SA Forum) is a consortium of several telecommunications and computing companies that defines standard middleware services for supporting and managing high-availability of applications. One of the most important SA Forum services is the Availability Management Framework (AMF), which is responsible for managing the availability of applications. To achieve this, AMF requires a configuration of the application, which consists of various entities organized according to specific rules and constraints defined in the AMF standard. Creating such configurations can be a difficult task due to the large number of possibly constrained entities involved. In this paper, we present a language and a supporting implementation that models the AMF domain to provide configuration designers with the tools needed to design, edit, and potentially analyze AMF configurations. Our language, called UACL (The UML-based AMF Configuration Language), is an extension of

UML through its profiling mechanism. UACL has been carefully designed to represent AMF concepts, their relations, and constraints. It has also been implemented using the IBM Rational Software Architect (RSA). We show the effectiveness of UACL in designing AMF configurations through a case study. We also report on our experience in designing this language and the challenges we encountered.

Keywords: Domain-specific modeling languages; UML profiles; High-Availability; Availability Management Framework Standard; Service Availability Forum.

1 Introduction

Service availability has long been an important software requirement, especially for safety-critical systems for which any service interruption may have fatal results, hence, the latest increase in attention to effective solutions that usually take the form of high-availability middleware for the development of highly available (HA) systems. The common practice is to have some sort of redundancy models so as failing nodes are detected and their workload is shifted to standbys. However, most existing solutions are proprietary and platform specific, which hinders the portability of the applications from one platform to another. To address this issue, many telecommunications and computing companies have joined forces in the Service Availability Forum (SA Forum) [SAF 2014], a consortium whose objective is to define open standard specifications in order to support the development and management of HA applications and to enable portability and reusability across different platforms. More specifically, the SA Forum standard service interfaces enable the use of Commercial-Off-The-Shelf (COTS) building blocks for an HA system, which results in the enhanced portability and flexibility of the components. Moreover, since the developers need only to focus on the application logics, SA Forum standards reduce the complexity of the application development.

The SA Forum specifications can be grouped into two categories:

- The Application Interface Specification (AIS) [SAF 2010a] which defines a set of services that are needed to fully support the high availability of the application's components.
- The Hardware Platform Interface (HPI) [SAF 2010b] which provides standard means to control and monitor hardware components.

There are several implementations of AIS provided by different groups. OpenAIS is an open source project that started at MontaVista [MontaVista 2015] as an implementation of AIS and later became part of Linux implementation and was licensed under the Revised BSD license [OpenAIS 2015]. Other projects such as Corosync also were derived from OpenAIS [Corosync 2015]. The most popular implementation AIS is OpenSAF which is supported by various telecommunication and software companies through OpenSAF Foundation [OpenSAF 2015].

The major implementation of HPI is OpenHPI, which is provided by the open source community [OpenHPI 2015].

Among the services defined in AIS, perhaps the most important one is the Availability Management Framework (AMF) [SAF 2010c], which is the middleware service that manages the high availability of the applications' services by coordinating the applications' redundant 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

The design of AMF configurations requires 1) the description of the software components provided by the vendor in an Entity Types File (ETF), which follows a standard format defined in [SAF 2010d], 2) the description of the deployment infrastructure, 3) the definition of the services to be provided, and 4) the required redundancy model. Such a design requires a good understanding of AMF concepts and their relations. This is a complex task due to the large number of entities and parameters that need to be designed and generated. There are also several constraints imposed by the AMF domain, which tend to crosscut various entities, making the configuration design process very complex. Also, the specifications are described in an informal way, leaving room for ambiguity and misinterpretations.

To address these issues, we have designed a precise and complete modeling language, called UACL[1] (a UML-based AMF Configuration Language) to enable the design and analysis of AMF configurations. Our language is a new domain-specific modeling language (DSML) tailored to capture the AMF domain's concepts and semantics.

Using UACL, one can benefit from the advantages of a domain specific modeling language (e.g. usability, the reuse and conservation of domain knowledge and the ease of communication) [Selic 2003] as well as from the advantages associated with UML, including its standard design notation and accessible tool support.

UACL is a UML profile [OMG 2012c] for the modeling of AMF domain concepts, their attributes, their relationships, and the domain specific constraints. It has been implemented using IBM Rational Software Architect (RSA) [IBM 2014]. We believe that it is an important contribution to the service high availability community since it aims to enable different activities, such as the design of configurations using domain concepts as first class artifacts, model driven generation of valid AMF configurations, the validation of third party AMF configurations.

In this paper, we also report on our experience in designing such a profile by discussing the challenges we have encountered and that we attribute mainly to three aspects: (a) the lack of a systematic approach for creating profiles, especially for the mapping of the domain concepts to the UML metamodel – in many situations, we have found that there were many alternatives and it was not always obvious which one to choose; (b) the use of Object Constraint Language (OCL) [OMG 2014a] which turned out to be problematic, since most of the domain constraints needed extensive OCL expressions that crosscut several domain contexts; and (c) the lack of flexible tool support. 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. It is worth noting that the work described in this thesis is part of a larger research project called MAGIC[1] —a collaboration between Concordia University and

[1] MAGIC (Modeling and Automatic Generation of Information and upgrade Campaigns for service availability). The acronym MAGIC is used throughout the description of UACL to reflect concepts and elements that have been newly introduced or further refined compared to standard specifications.

Ericsson— and the results of this thesis are being used in other MAGIC research streams. The term MAGIC is used throughout the profile.

The remaining part of this paper is structured as follows. In Section 2, we introduce the main concepts in the AMF specification. In Section 3, we present the related work where we reviewed different UML profiles relevant to our domain. In Section 4, we describe the methodology 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 theUML metamodel. In Section 5, we present the implementation of UACL and a case study, followed by a discussion of the challenges in Section 6. We conclude the paper in Section 7.

2 AMF Configurations

AMF [SAF 2010c] 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. In order to manage the services, AMF requires a configuration that specifies the organization and the characteristics of the entities under its control. These entities model the service providers, the provided services, and the deployment information. An AMF configuration consists of two different sets of concepts: AMF entities and the associated AMF entity types. AMF entities are categorized into different logical entities representing services and service providers' resources. The basic entities are called Components. Components represent hardware or software resources capable of supporting the workload of the application services. Such a workload is referred to as a Component Service Instance (CSI). For example, an instance of MySQL server could be a component called MySQL_1 which is capable of supporting a specific set of clients. The IP addresses of these clients form the description of the workload for this specific instance of MySQL component which is captured through a CSI (MySQL_1_CSI). Components are aggregated into Service Units (SU), logical entities representing the basic redundancy unit for AMF. The aggregation of components (e.g. MySQL_1, Driver Manager_1, and JDBC Connector_1) enables the combination of their functionalities to form higher level services. More specifically, the workloads of the components of an SU are aggregated into a Service Instance (SI), which represents the aggregated workload assigned to the SU. An SI also represents the combined higher level service of the collaborating components within the SU. In our example, the combination of MySQL_1, Driver Manager_1, and JDBC Connector_1 forms a database tier (SU) which can provide service for software systems, developed using Java based technologies.

Table 1 Description of AMF entities

Entity	**Description**
Component	Represents a hardware or software resource that can provide a service.
Component Service Instance (CSI)	Represents a service workload assigned to a component. AMF assigns a workload to a component at run-time.
Service Unit (SU)	Groups a set of components in a node, which collaborate to provide services.
Service Instance (SI)	Represents a service an SU can provide. It is an aggregation of CSIs.
Service Group (SG)	It consists of a set of redundant SUs. It protects the SIs assigned to these SUs.
Application	Represents a logical entity that contains one or more SGs and combines the individual functionalities of the constituent SGs to provide a higher-level service.
Node	Represents a computational resource for the deployment of artefacts.
Node Group (NG)	A set of AMF nodes that can be configured for an SG or an SU.
Cluster	Represents the aggregation of the complete set of AMF nodes in an AMF configuration.

SUs are further grouped into Service Groups (SG) in order to protect a set of SIs by means of redundancy. SGs are characterized by redundancy models. AMF supports the No-Redundancy, 2N, N+M, N-Way and N-Way-Active redundancy models. These redundancy models differ on the number of SUs that can be active and on standby for the SIs and on how these assignments are distributed among the SUs. In the 2N redundancy model, one SU is active for all the SIs protected by the SG and one is on standby for all the SIs. In the N+M model, N SUs support the active assignments and M SUs support the standbys. N+M allows at the most one active and one standby assignment for each particular SI. An SG with N-Way redundancy model contains N SUs. Each SU can have a combination of active and standby assignments. However, each SI can be assigned active to only one service unit while it can be assigned as on standby to several service units. An SG with the N-Way-Active redundancy model has N SUs which are assigned only as active. It has no SU assigned as on standby. Furthermore, each of the SIs protected by this SG can be assigned to more than one SU. In contrast to N-Way-Active, in the No-Redundancy redundancy model each SI is assigned to at most one SU and each SU can protect at most one SI. An AMF application is composed of one or more service groups. Each SU is deployed on an AMF node and the set of all AMF nodes forms the AMF cluster. Table 1 summarizes the AMF entities and their descriptions.

AMF entity types are used to define the common characteristics among multiple instances of the same type. For example, a component type can define all the characteristics/attributes of the MySQL server which are common among all instances of this server. In AMF, all entities except for the deployment entities (i.e., node, NG, and cluster) have a type. The types are derived from a vendor's description of the application in the ETF [SAF 2010d].

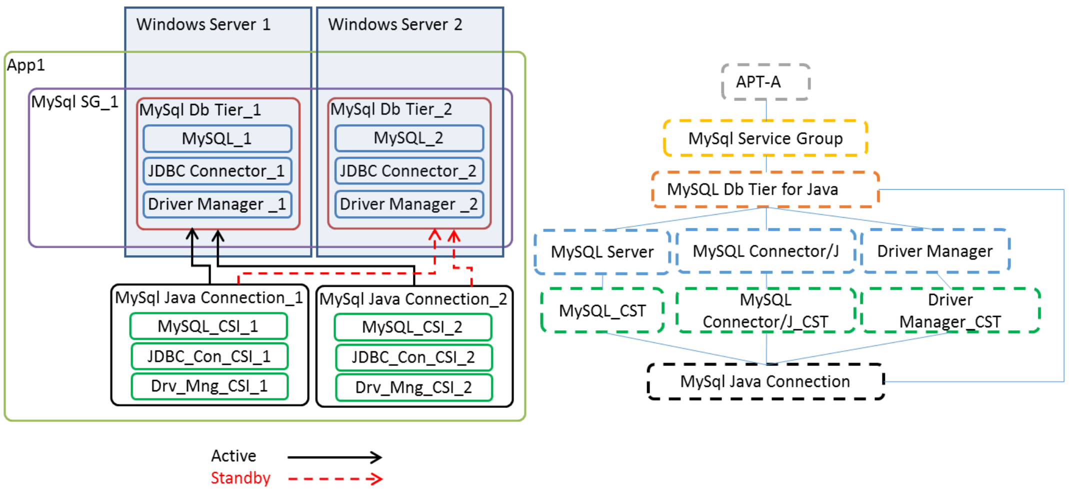


Figure 1 An example of AMF configuration

Figure 1 shows an example of an AMF configuration. In this example, a cluster is composed of two nodes (Windows Server 1 and Windows Server 2). It hosts an application consisting of one SG (MySql SG_1) protecting two SIs (MySql Java Connection_1 and MySql Java Connection_2) in a 2N redundancy model. The MySql SG_1 consists of two SUs, MySql Db Tier_1 and MySql Db Tier_2, each being composed of three components. Although shown in Figure 1, the distribution of the active and standby assignments is not part of the configuration as defined by AMF, since this is decided by AMF at run-time. The types, part of the configuration, are shown in the right hand side of the figure. The relationship between the type entities and the entities presented in the configuration are as follows: MySQL_1 and MySQL_2 are from the Component Type MySQL Server. JDBC Connector_1 and JDBC Connector_2 are from MySQL Connector/J. Both the SUs are represented by the same SUType called MySQL Db Tier for Java. MySql SG_1 and App1 are from the type MySql Service Group and APT-A, respectively. At the service level, both SIs (MySql Java Connection_1 and MySql Java Connection_2) are from the type MySql Java Connection while the CSIs are from three different types. More specifically, MySQL_CSI_1 and MySQL_CSI_2 are of the type MySQL_CST, JDBC_Con_CSI_1 and JDBC_Con_CSI_2 are from the type MySQL Connector/J_CST and finally, Drv_Mng_CSI_1 and Drv_Mng_CSI_2 are from Driver Manager_CST.

3 Related Work

There are several UML profiles (some of them standardized) that model concepts such as components and services, which are also key concepts in AMF. The question is therefore: Do we need to define a UML profile from scratch or simply reuse (or extend) an existing one? This question has always been a matter of debate since each option has its own benefits and disadvantages. Unfortunately, there is no formal process of finding out whether it is better to extend an existing profile or to create a new one. In this section, we present a brief review of related UML profiles together with the rationale supporting our decision to create a new profile, instead of extending an existing one.

There are three main UML profiles defined and standardized by OMG [OMG 2014b] which represent some concepts that are also found in AMF. These profiles are: SPT [OMG 2005], MARTE [OMG 2011], and the UML profile for QoS&FT [OMG 2008]. There are also other profiles related to the AMF concepts, namely the DAM Profile [Bernardi 2011] and the profile introduced in the HIDENETS project [Kövi 2007]. These two profiles are to some extent either extending or reusing parts or all of one of the aforementioned OMG profiles.

The UML SPT profile [OMG 2005] focuses on the properties related to the modeling of time and time-related aspects such as the concept of clocks, the key characteristics of timeliness, performance, and schedulability. Despite the fact that the authors introduce a set of sub-profiles in order to extend the core of SPT, which is the general resource modeling framework and which can be used by other profiles for availability analysis, there are no specific means for modeling availability related issues such as redundancy models in SPT. Consequently, when reusing SPT, one should define all necessary constructs for AMF configurations. However, basing this definition on SPT's abstract syntax may complicate the design process of our language by imposing extra constraints inherited from SPT and which are not related to the AMF domain.

The MARTE profile [OMG 2011], an extension of SPT, defines a package for Non-Functional Properties (NFP) which supports new user-defined NFPs for different specialized domains [OMG 2011]. It also defines a package for the purpose of analysis called the Generic Quantitative Analysis Modeling (GQAM). UML profile for QoS&FT defines a general QoS catalogue including a set of general characteristics and categories [OMG 2008]. In particular, this profile defines a package for availability related concepts, focusing on representing attributes, such as mean time to failure that can be used to measure the availability of services. Similarly to SPT, these profiles omit to model key concepts of high-availability including the redundant structures which play a critical role in highly available systems. Extending them has the same drawbacks as for SPT.

Both NFA and GQAM packages (from the MARTE Profile) have been reused in the design of the Dependability Analysis Modeling (DAM) profile (an extension to MARTE) in order to enhance modeling facilities for the purpose of analysing dependability [Bernardi 2011]. In the DAM profile, the building blocks of a system are limited to components (DaComponent mapped to MARTE::GRM::Resource) and services (DaService mapped to MARTE::GQAM::GaScenario). The DAM profile does not model many key AMF concepts including proxies, service units, component service instance, and so on. In addition, the concept of services in the DAM profile describes the service itself, whereas in AMF, the service describes the workload to be assigned to a service provider at run-time. Adapting the DAM profile to AMF would require as much effort as creating a new profile, if not more. We have decided to opt for the development of a new profile to avoid being restricted with the DAM semantics.

The HIDENETS profile [Kövi 2007] was introduced to model software that runs on the HIDENETS platform. The HIDENETS middleware provides a basis for mobility-awareness and for the distribution of applications. The designers of this profile have reused several standard UML profiles such as SPT, QoS&FT, SysML [OMG 2012b], AUTOSAR Profile [AUTOSAR 2006], and MAM-UML [Belloni 2004]. In addition, the HIDENETS profile is compliant with the AMF specification [SAF 2010c].

HIDENETS utilizes AMF concepts using the façade design pattern. In this profile the authors provided a general AMF façade, the point through which the middleware interacts with the AMF elements. In other words, a set of well-designed AMF compliant components can interact with the HIDENETS middleware through the AMF façade. HIDENETS profile does not model AMF configuration concepts and therefore, cannot be used to fulfill our goal of specifying and analyzing AMF configurations.

The work described in [Szatmári 2008] is probably the work most related to this paper. The authors introduced an MDA (Model-Driven Architecture) approach for the automatic generation of SA Forum compliant applications. They have introduced a metamodel for SA Forum compliant applications based on the class diagram introduced for AMF configurations in AMF specification [SAF 2010c]. Based on their work, an application should be first modeled using their metamodel and then, by using an MDA approach, the SA Forum compliant APIs are added and the source code for the application is generated. The authors have also provided the configuration for the subject application. Although this work concentrates more on application development, it also generates the required AMF configuration for the application. This work is limited to in-house software development and does not handle third party software. Moreover, the software development and configuration design are handled at the same time which does not leave much flexibility for configuration and deployment time. After careful examination of the profile described in [Szatmári 2008], we found that it does not take into account many AMF domain constraints. Capturing and specifying the constraints is an important step in the definition of a UML profile. In particular, in complex domains such as AMF configurations, class diagrams alone are not sufficient to express all domain specific concepts and their relations. A nonconstrained domain model does not guarantee a profile that can be used to validate third party configuration written in this profile, which is one of the objectives of our work. From the implementation point of view, one of the goals of our work is to develop a CASE tool to support different activities. In the case of extending existing profiles, we need to have access to their implementation such as the OMG XML Metadata Interchange (XMI) [OMG 2007a] format that serializes the profile model. We found that the non-standard profiles do not provide open access to their implementation. Although the implementation is available for some of the standard OMG profiles (e.g., MARTE), 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 the newly defined constraints of the new language may require the evaluation of several constraints of the referred profile.

In the work by Turenne et al [Turenne 2014a and Turenne 2014b], the authors improve on their previous work [Kanso 2008 and Kanso 2009], by adding a model for the description of software components specified in ETF. Their model captures the description of the software components which is used to generate the configuration when these components are deployed and not the AMF configuration itself.

In the area of standardization, Lightweight Fault Tolerance (LWFT) [OMG 2012a], which extends the Fault Tolerance (FT) CORBA [OMG 2010] specification, provides an availability management solution to support real-time applications. The applications however, need to follow CORBA architecture. In [Toeroe 2012], the authors discuss the difference between CORBA based solutions and SA Forum in detail.

4 Designing the Profile

Unfortunately, there has been little material written on how to create UML profiles. As a result, most existing UML profiles have been defined in an ad hoc manner, ending up being either technically invalid, contradicting the UML metamodel, or of poor quality [Selic 2007, Lagarde 2007, Lagarde 2008]. In the work presented in this paper, we followed the approach proposed by Selic in [Selic 2007]. This approach consists of the following two steps:

- Specifying the domain model (or domain metamodel): The domain model specifies the concepts that pertain to the domain specific modeling language and how these concepts are represented. The output of this phase consists of fundamental language constructs, relationships between domain concepts, and any constraints imposed by the domain. The concrete syntax or the notation used to render these concepts, and the semantics of each language construct are also described at this stage.
- Mapping the domain model to the UML metamodel: This step consists of identifying the most appropriate UML base concepts for each domain concept specified in the previous step. In this step, the profile designer needs to choose the base UML metaclass which is semantically closest to the semantics of the domain concept. Moreover, the constraints, attributes, and related associations of the selected meta-elements should be verified in order to prevent the contradiction of the domain concepts.

4.1 Defining the AMF domain model

The AMF domain model has been developed by studying the AMF specifications and through constant interactions with an AMF domain expert. A class diagram describing the different types, entities and some of their relationships is provided in the standard. This class diagram is kept simple for the purpose of administration and runtime management. It is not appropriate for the purpose of configuration design and validation as it does not capture all the properties and constraints for an AMF configuration.

The AMF domain elements are modeled as UML classes and the relationships among them are modeled through different types of UML relationships. The well-formedness rules of the AMF domain model elements have been specified using OCL. Figure 2 represents the process of specifying the AMF domain model.

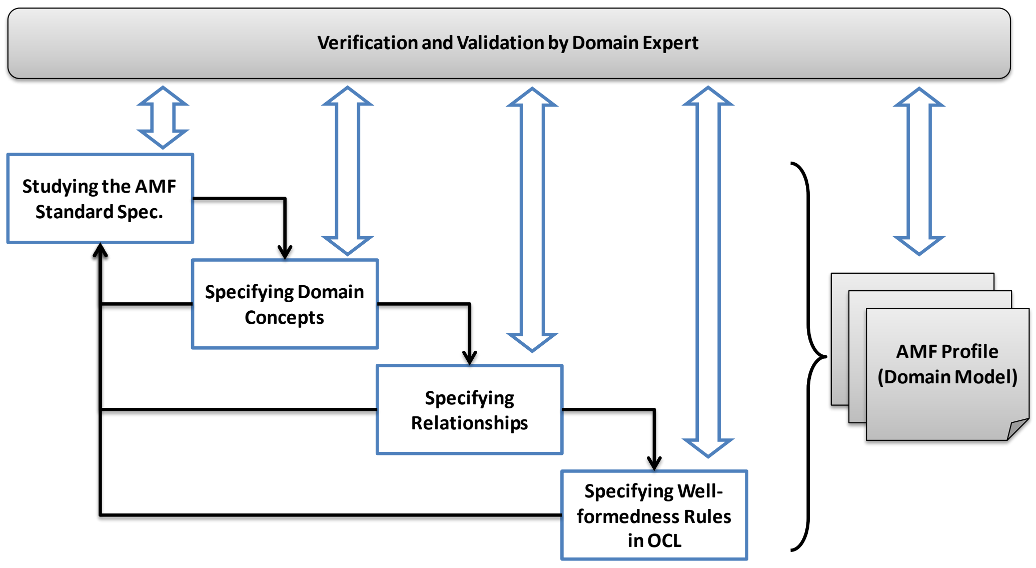

Verification and Validation by Domain Expert
Studying the AMF Standard Spec.
Specifying Domain Concepts
Specifying Relationships
Specifying Well-formedness Rules in OCL
AMF Profile (Domain Model)

Figure 2 Domain Modeling Process

4.1.1 AMF domain concepts and relationships

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 those representing 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 summarize the key AMF domain model elements and their relations.

In the standard specification, we can distinguish different AMF component categories along four orthogonal criteria: locality, service availability awareness (SA-awareness for short), containment, and mediation (see Figure 3). 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 instead use an intermediate environment, referred to as the container component, like a virtual machine (for example, to support Java-like programs). SA-aware components can be categorized further into proxy and container components. Proxies are used to give AMF control over hardware or legacy software, called proxied components. Container components allow AMF to control 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 always proxied to be controlled by AMF. These categories are captured in Figure 3 differently from the standard specification where they are distinguished with attributes.

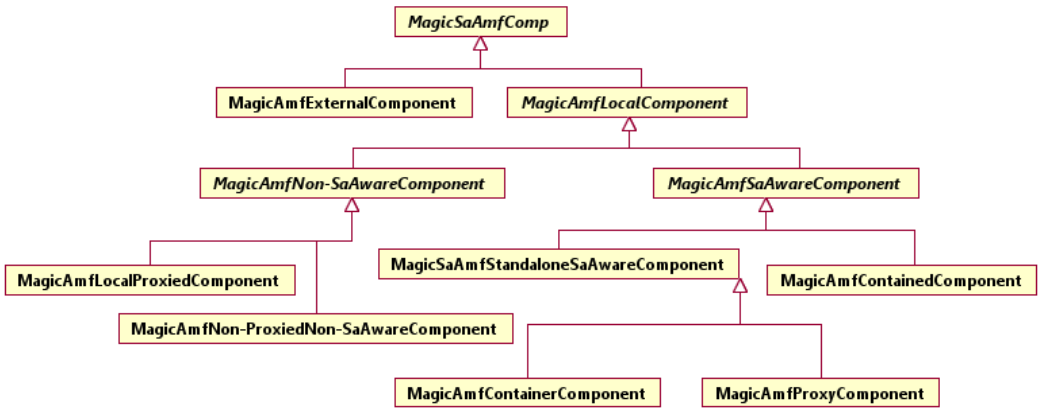

MagicSaAmfComp
MagicAmfExternalComponent
MagicAmfLocalComponent
MagicAmfNon-SaAwareComponent
MagicAmfSaAwareComponent
MagicAmfLocalProxiedComponent
MagicSaAmfStandaloneSaAwareComponent
MagicAmfContainedComponent
MagicAmfNon-ProxiedNon-SaAwareComponent
MagicAmfContainerComponent
MagicAmfProxyComponent

Figure 3 AMF Component Categories

Unlike the component classification, the component types do not take into consideration the locality criterion. This is because the component type cannot specify whether its components have to be located outside or inside the AMF cluster. In fact, a component type can specify whether its implementation captures 1) the APIs required to interact with AMF or 2) the necessary states for being proxied by another component type. As a result, the component type class models the types of the SA-aware components, 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, the standalone component type is further specialized into the proxy component type and the container component type which are the types of the proxy and container components, respectively. Figure 4 presents the different categories of AMF component types.

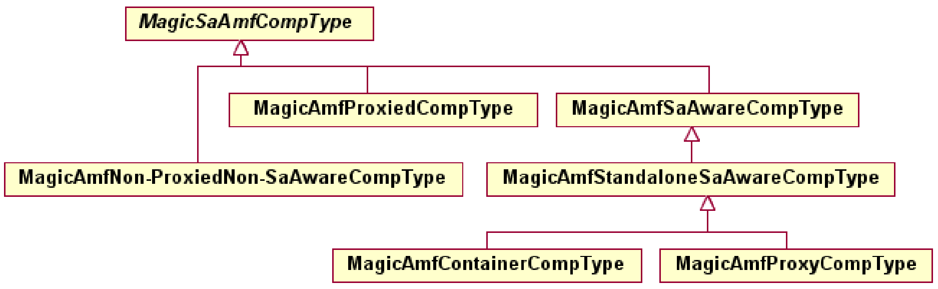

MagicSaAmfCompType
MagicAmfProxiedCompType
MagicAmfSaAwareCompType
MagicAmfNon-ProxiedNon-SaAwareCompType
MagicAmfStandaloneSaAwareCompType
MagicAmfContainerCompType
MagicAmfProxyCompType

Figure 4 AMF Component Type Categories

To provide a higher level service, components are grouped into SUs. We distinguish between local and external SUs (see Figure 5) depending on whether or not they contain local or external components. SUs are organized into 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 Figure 5). Each redundancy model has different characteristics and therefore, different sets of attributes and relationships with other elements (e.g. SUs). The specialisation of SGs based on the redundancy models facilitates the design and analysis of AMF configurations by reducing the complexity of the configuration. 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 SUTypes. 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 CSI and SI entities are captured in our domain model as shown in Figure 6.

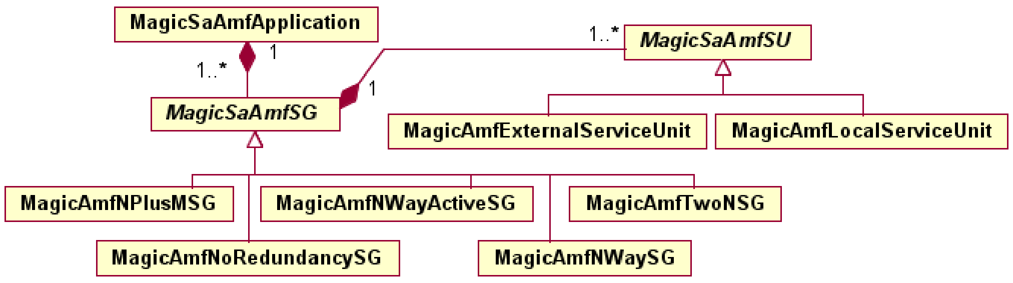


Figure 5 Service Unit and Service Group Categories

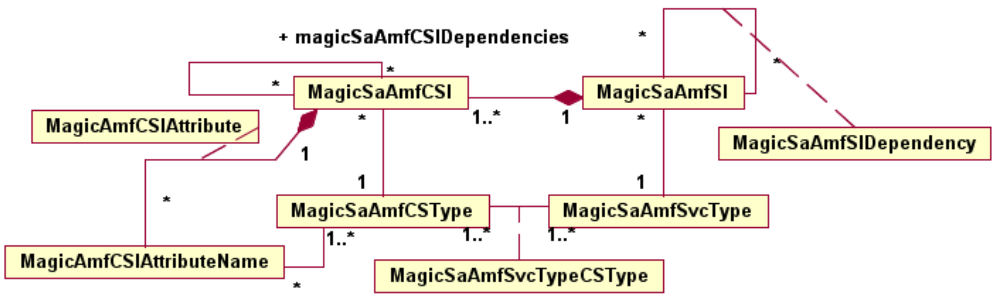


Figure 6 Component Service Instance and Service Instance

The AMF cluster, the AMF node and the node group represent part of our model for the deployment entities (see Figure 7). An AMF cluster is the complete set of AMF nodes in the AMF configuration. An AMF node represents a cluster node that can host AMF entities A node group represents a set of AMF nodes and is used for the deployment of local SUs and SGs. More specifically, each local SU can be configured to be deployed on one of the nodes of a node group and AMF decides the hosting node at runtime.

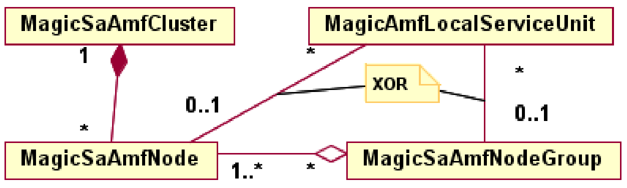


Figure 7 AMF Nodes, Node Groups, and Cluster

A detailed description of all domain elements, their attributes and relationships is presented in [Salehi 2012].

4.1.2 Specifying Well-formedness Rules

We used OCL to describe the constraints on the AMF domain model elements. We have categorized the well-formedness rules into three different groups: 1) configuration attributes, 2) structural constraints, and 3) constraints for ensuring the protection of services that a configuration claims to achieve. The first two groups can be seen as syntactical constraints while the last group of constraints is more about the semantic correctness of the configuration. The complete list of constraints is presented in [Salehi 2012]. In this paper, we provide examples from each category of constraints.

4.1.2.1 Configuration Attributes Well-formedness Rules

As discussed earlier in this paper, one of the main reasons for the complexity of AMF configurations is the large number of configuration attributes, parameters to be considered and the related constraints. These constraints among the attributes form the category of configuration attributes well-formedness rules. For instance, among the attributes of a component type, the *magicSaAmfCtDefDisableRestart* attribute specifies whether the restart recovery action is disabled by default for the components of this component type and the *magicSaAmfCtDefRecoveryOnError* attribute specifies the default recovery action that should be taken by the middleware for the components of this type in case of a failure. Based on the standard, for a certain component type, if the *magicSaAmfCtDefDisableRestart* is configured to *true*, then the attribute *magicSaAmfCtDefRecoveryOnError* must not be set as *SA_AMF_COMPONENT_RESTART* or *SA_AMF_NO_RECOMMENDATION*. This constraint is captured in the domain model and specified in OCL as:

```
  context MagicSaAmfCompType
  inv:
  magicSaAmfCtDefDisableRestart = true) implies
(magicSaAmfCtDefRecoveryOnError <> SA_AMF_COMPONENT_RESTART
         and
 magicSaAmfCtDefRecoveryOnError <> SA_AMF_NO_RECOMMENDATION)
```

Several other restrictions on the attributes defined in the AMF specification are, however, complex and not straightforward to express. This complexity stems from the fact that, in an AMF configuration, these requirements crosscut entities and concepts from different levels. This is the case, for example, when a constraint involves different concepts such as the component capability and the redundancy model.

Figure 8 depicts part of the AMF domain model which represents the relationships of the CSType with the component type and the component. Both relationships are represented through association classes. The AMF domain specification states that: *for all CSTypes which are provided by a component, the value of the attribute magicSaAmfCompNumMaxActiveCSIs in the association class between component and CSType should be lower than or equal to the value of the attribute magicSaAmfCtDefNumMaxActiveCSIs which is located in the association class between the CSType and the component type of that component.* This is an example of a cross-context constraint which has been captured in the domain model and specified in OCL as follows:

```
context MagicSaAmfComp
inv:
self.magicSaAmfCompCsType->
forAll(compcst|compcst. magicSaAmfCompNumMaxActiveCSIs <=
       self.magicSaAmfCompType.magicSaAmfCtCsType ->
         select(ctcst | ctcst.magicSafSupportedCsType =
                        compcst.magicSafSupportedCsType)->
       asSequence.at(1). magicSaAmfCtDefNumMaxActiveCSIs)
```

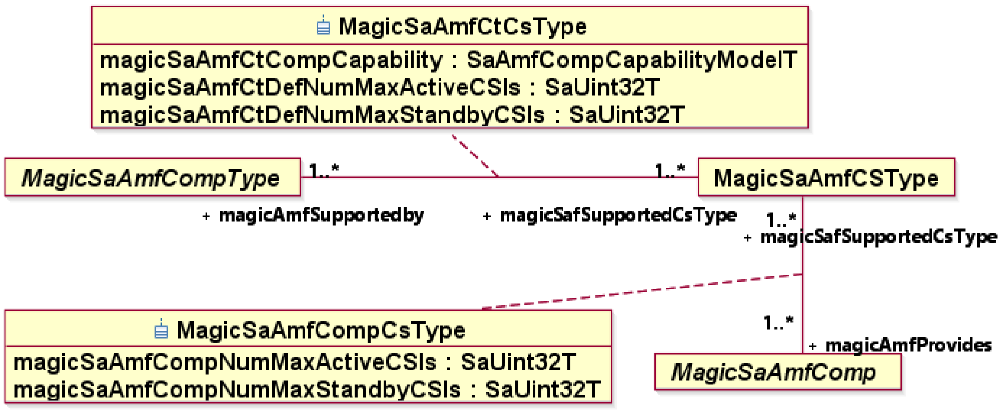

MagicSaAmfCtCsType
magicSaAmfCtCompCapability : SaAmfCompCapabilityModelT
magicSaAmfCtDefNumMaxActiveCSIs : SaUint32T
magicSaAmfCtDefNumMaxStandbyCSIs : SaUint32T
MagicSaAmfCompType
1..*
1..*
MagicSaAmfCSType
+ magicAmfSupportedby
+ magicSafSupportedCsType
1..*
+ magicSafSupportedCsType
MagicSaAmfCompCsType
magicSaAmfCompNumMaxActiveCSIs : SaUint32T
magicSaAmfCompNumMaxStandbyCSIs : SaUint32T
1..*
+ magicAmfProvides
MagicSaAmfComp

Figure 8 Relationship of CSType with component and component type

4.1.2.2 Structural Well-formedness Rules

The elements of the AMF configurations are strongly related, resulting in a complicated organization of configuration elements. More specifically, the configuration entities and entity types form two levels of abstraction which need to be compliant with each other. In addition, in each level there are nested relationships among the elements (e.g. SG groups SUs and each SU groups components). Therefore, the second category of well-formedness rules is concerned with ensuring the structural consistency of the configuration with respect to the standard. As an example of a structural 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 (see Figure 9). We can therefore easily capture this restriction in OCL as follows:

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

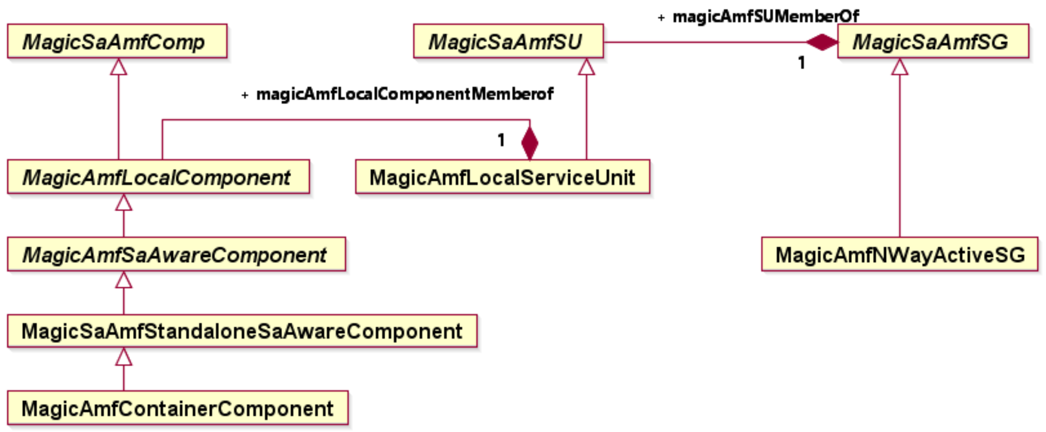


Figure 9 Series of relationship between Container Component and N-Way-Active Service Group

4.1.2.3 Service Protection Constraints

A configuration is semantically valid only if it is capable of providing and protecting the services as required and according to the specified redundancy model. More specifically, given a set of SUs grouped in an SG, one needs to ensure that the set of SUs is capable of handling the SIs configured for the SG. We formalized the service protection problem for all the redundancy model using higher order logic (HOL). Ensuring this (referred to as SI protection problem) requires the exploration of all possible SI-SU assignments. In some cases it is necessary to consider different combinations of SIs, which make the problem complex in most redundancy models. For instance, the problem has combinatorial aspects in N-Way and N-Way-Active, and N+M redundancy models where the SIs can be assigned to more than one SU, so there are many valid assignment combinations. For these cases the problem is NP-hard [Salehi 2009]. We tackled the problem by providing the necessary and sufficient conditions for ensuring the SI protection for each redundancy model. In the case of the 2N redundancy model and the No-Redundancy redundancy model, we have been able to characterize the necessary and sufficient conditions for the general case and specified this in the form of well-formedness rules in OCL. For example the conditions for the 2N redundancy model are summarized as:

A service unit in the MagicAmfTwoNSG should be able to be active for all service instances protected by the service group and *a service unit in the MagicAmfTwoNSG should be able to be on standby for all service instances protected by the service group* (see Figure 10).

The OCL constraints (for simplicity we present the constraint for local service units) specifying the well-formedness rule for the active/standby assignment of 2N redundancy model is:

```
  context MagicAmfTwoNSG
  inv:
 (self.magicAmfSGGroups->forAll(su |
    su.oclIsTypeOf(MagicSaAmfLocalServiceUnit))
    implies
  (su.magicSaAmfSUType.magicSaAmfSutProvidesSvcType->
    forAll(svct | svct.magicSaAmfSvcTypeCSType.
     magicSafMemberCSType->
      forAll(cst | su.magicAmfSUMemberOf.
        magicAmfSGProtects ->
         iterate(si; b:integer = 0 | si.magicAmfSIGroups->
           select(csi |
             csi.magicSaAmfCSType = cst)->size()+b) <=
               su.magicAmfLocalServiceUnitGroups->
                iterate(c ; a:integer = 0|
                 c.MagicSaAmfCompCsType->
                  select (compcst | compcst.
                    magicSafSupportedCsType = cst)->
                     asSequence.at(1).
                      magicSaAmfCompNumMaxActiveCSIs+a)))))
  and

(self.magicAmfSGGroups->forAll(su |
    su.oclIsTypeOf(MagicSaAmfLocalServiceUnit))
    implies
  (su.magicSaAmfSUType.magicSaAmfSutProvidesSvcType->
    forAll(svct | svct.magicSaAmfSvcTypeCSType.
     magicSafMemberCSType->
      forAll(cst | su.magicAmfSUMemberOf.
        magicAmfSGProtects ->
         iterate(si; b:integer = 0 | si.magicAmfSIGroups->
           select(csi |
             csi.magicSaAmfCSType = cst)->size()+b) <=
               su.magicAmfLocalServiceUnitGroups->
                iterate(c ; a:integer = 0|
                 c.MagicSaAmfCompCsType->
                  select (compcst | compcst.
                    magicSafSupportedCsType = cst)->
                     asSequence.at(1).
                      magicSaAmfCompNumMaxStandbyCSIs+a)))))
```

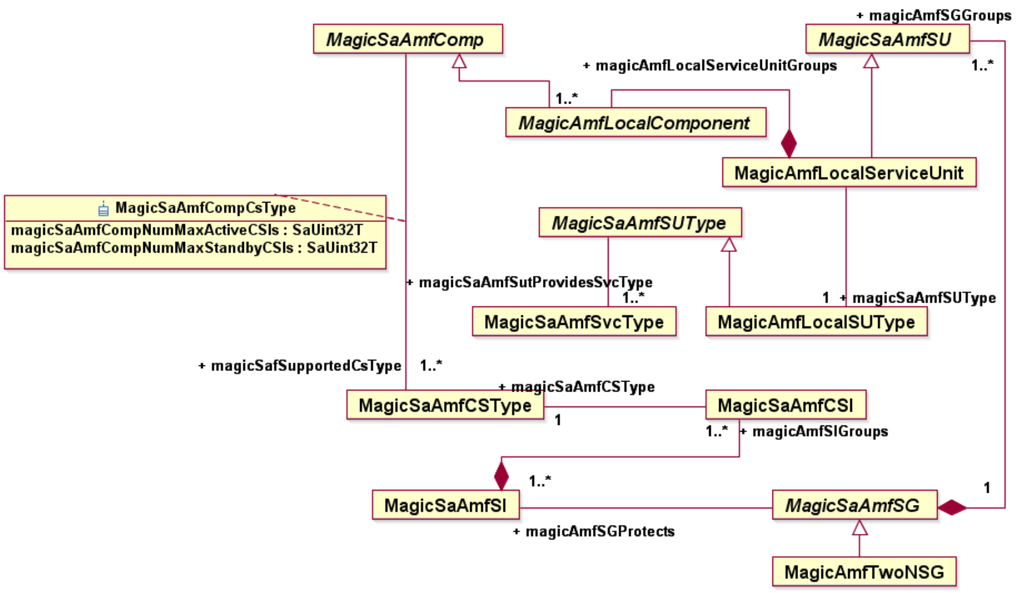


Figure 10 Partial view of the domain model involved in 2N Service Group well-formedness rule

For overcoming the complexity in the case of the N+M, the N-Way-Active, and the N-Way redundancy models, we have characterized some specific cases, where the necessary and sufficient conditions can be checked efficiently and specified them as OCL constraints. The details of the formal description of the SI protection problem as well as the complexity analysis and the proposed solutions are presented in [Salehi 2009].

4.2 Mapping the AMF Domain Model to the UML Metamodel

Although in [Selic 2007], the author proposes the separation of the domain modeling phase and the mapping phase, he does not provide any guidelines for this mapping, which is perhaps the most challenging activity in defining a well-formed UML profile. The International Telecommunication Union (ITU) also provides guidelines [ITU 2007], which mainly focus on the profile document, common conventions, and recommendations on how to present the text and notations but not on the mapping phase. A lack of a systematic approach (or at least insightful guidelines) for selecting the most suitable metaclasses makes this phase dependent on the experience of the profile's designer. Other studies [Lagarde 2007, Lagarde 2008] propose patterns which are based on a few types of relationships that may exist between domain elements and the corresponding metaclasses. However, these guidelines focus on specific scenarios and do not provide a general solution to the mapping problem. In other words, there is no “ready to use” solution that addresses the general issue of selecting the most appropriate UML metaclass for a specific domain element.

During the design of our profile, we have selected the UML metaclasses that carry semantics similar to the domain concepts being represented. As such, the newly defined stereotypes must neither contradict nor violate the UML metamodel. In the presence of multiple candidates, we favoured the metaclasses that permitted the reuse of as many UML relationships between the stereotyped elements as possible. Reusing the associations among the metaclasses decreases the complexity of the design. Hence, if it is necessary to have a relationship between two stereotypes, it is better to reuse (if possible) the existing relationships between the corresponding metaclasses. We also opted for the metaclasses that minimized the number of constraints needed to constrain the UML metamodel elements (i.e., to restrict the stereotyped UML metaclasses so as to have them behave according to the rules imposed by the domain). A large number of constraints is an indication that the selected metaclasses might not be the most suitable ones.

One needs to proceed step by step through the full set of domain concepts (specified as classes in the domain model), identifying the most appropriate UML base concepts for each of them. In this step, the objective is to find the UML base concept (UML metaclass) which is conceptually and semantically similar to each domain concept. The output of the mapping phase is a set of new stereotypes and the UML metaclass from which each stereotype is derived. It is important to mention that, since UML 2.0 supports the inheritance relationship between stereotypes, not all domain concepts need to be directly derived from a corresponding UML metaclasse. Some of them will be derived from the newly created stereotypes. Figure 11 illustrates the process of mapping the domain model to the UML metamodel, the definition of the concrete syntax for the language, and the specification of the metamodel level constraints. We follow systematically the steps of this process to guarantee the quality of our profile and the remainder of this section is dedicated to presenting each step in detail.

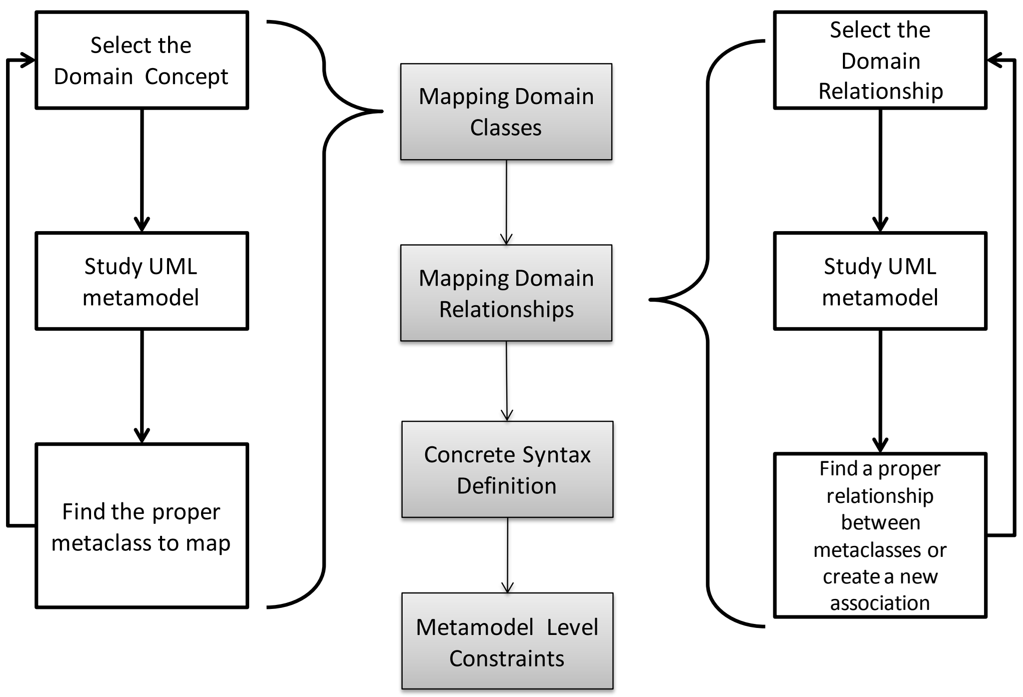

Select the Domain Concept
Study UML metamodel
Find the proper metaclass to map
Mapping Domain Classes
Mapping Domain Relationships
Concrete Syntax Definition
Metamodel Level Constraints
Select the Domain Relationship
Study UML metamodel
Find a proper relationship between metaclasses or create a new association

Figure 11 The process of mapping to the UML metamodel and concrete syntax definition

4.2.1 *Mapping AMF Domain Model Concepts to UML Metaclasses*

For each stereotype a suitable metaclass is presented (see Appendix A). This selection has been made by mainly considering the semantic alignment of the domain concepts with respect to UML metaclasses. However, the first choice might not be the most appropriate (most semantically aligned) one and further investigation is necessary. More specifically, after finding the candidate metaclasses for each domain concept, two different scenarios may occur:

1. The candidate metaclass semantically appears to be appropriate: in this case it is always beneficial to look at the child metaclasses specializing the candidate metaclass. In other words, since the child metaclasses specify more features, we may find them semantically more accurate for aligning with the description of the domain concept.
2. The candidate metaclass turns out to have features which are semantically too restrictive compared to the description of the domain concept. In this case, one should consider the parent metaclass which technically has fewer features.

These guidelines highly support the semantic alignment of the domain concepts with respect to the UML metamodel. Considering these guidelines in the rest of this section, we present the complete set of stereotypes defined in our profile for each domain concept. For each stereotype we also present the most suitable UML concept for mapping. In addition, the rationale behind the selection of each UML metaclass is presented. It is worth noting that, for most domain concepts, there is more than one UML metaclass as an alternative for mapping and the most appropriate alternative has been selected after an extensive study of the UML metamodel and the AMF domain model.

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"* [OMG 2007b]. Therefore, we mapped the AMF component to the 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 metaclass.

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 the individual functionalities of these components to provide a higher level service. From this perspective, one could see an SU as a service provider, similar to a component, but at a higher level of abstraction. We therefore decided to map the SU to the UML Component metaclass as well. The stereotype <<*MagicSaAmfSU*>> is used to represent an SU. Local and external SUs are represented using the stereotypes <<*MagicAmfLocalServiceUnit*>> and <<*MagicAmfExternalServiceUnit*>>.

Service Group

One of the key characteristics of a SG is the grouping of SUs. Given the fact that in UML *"a package is used to group elements, and provides a namespace for the grouped elements"* [OMG 2007b], it may appear that the metaclass Package could be a suitable base class for an SG. However, in addition to its ability to group

SUs, an SG also ensures the availability of services by means of redundancy models for a certain set of SIs (assigned to the SUs grouped by the SG). Moreover, UML Component can liberally provide any kind of service. Consequently, we can consider the protection of SIs as a sort of service that is provided by the SG through importing SUs in its namespace. Therefore, similar to an SU, an SG can map to the UML Component metaclass. Considering the fact that the Component metaclass also has a grouping capability, it is the most appropriate candidate base class for the SG.

There are different categories of SGs based on their redundancy model, and so, for each category we have introduced a stereotype. The topmost stereotype (<<*MagicSaAmfSG*>>), however, has been mapped to the UML Component metaclass.

Application

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

Component Service Instance (CSI)

In the UML specification, a Classifier is an abstract metaclass which is a namespace whose members can include features. A BehavioralClassifier is a specific type of Classifier that may have an interface realization [OMG 2007b]. Since we can consider CSIs as realizations of services which AMF dynamically assigns to components in terms of workload, BehavioredClassifier could be a good candidate for CSI. However, a CSI is the description of the characteristics of the workload which will be assigned to the component at run-time and not the description of the service itself. Therefore, BehavioredClassifier 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"* [OMG 2007b], and thus, the metaclass Class is semantically closer to a CSI. As a result, we have used the metaclass Class as a base class for the stereotype that has been defined for CSI (<<*MagicSaAmfCSI*>>).

Service Instance (SI)

An SI is an aggregation of all component service instances (CSIs) to be assigned to the individual components of the SU in order for the SU to provide a particular service. In fact, semantically, an SI shares most of the characteristics of the CSI but at a higher level of abstraction. Consequently, similar to CSI, the metaclass Class can be used as a base class for the stereotype defied for an SI (<<*MagicSaAmfSI*>>). The only difference existing between the two is that the SI is capable of grouping a set of CSIs. This capability is also captured by the metaclass Class in UML due to the existence of an inheritance relationship between the metaclass Class and the metaclass Classifier.

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 metaclass Node since, similar to AMF, a node in UML *"is a computational resource upon which artefacts may be deployed for execution"* [OMG 2007b]. We created the stereotype <<*MagicSaAmfNode*>> to refer to an AMF node.

Cluster and Node Group

Based on the UML specification, *"a package is used to group elements, and provides a namespace for the grouped elements"* [OMG 2007b]. On the other hand, the complete set of AMF nodes in the AMF configuration defines the AMF cluster. The role of an AMF cluster and nodegroup is the grouping of different AMF nodes. Therefore, the metaclass Package seems to be the most appropriate base class for the AMF cluster and nodegroups. The stereotypes <<*MagicSaAmfCluster*>> and <<*MagicSaAmfNodeGroup*>> are used to refer to these two entities.

AMF Entity Type elements

In general, the type entity describes the characteristics and features common to all entities of this type. 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 SGType is a generalization of similar SGs that follow the same redundancy model, provide similar availability, and are composed of units of the same SUTypes. Considering the fact that, in UML, the metaclass Class describes a set of objects that share the same specifications of features, constraints, and semantics [OMG 2007b], it can be used as a base class for all AMF entity types.

Appendix A presents the summary of the stereotypes defined for AMF entities and entity types as well as the graphical syntax of our language for each stereotype.

4.2.2 Mapping the AMF relationships to the UML Metamodel

We distinguish six different 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 an SG and SIs in order to protect the services they represent.
- Deploy: It represents the installation 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 nodegroup or cluster. It is non-exclusive, as a node may be member of different node groups.

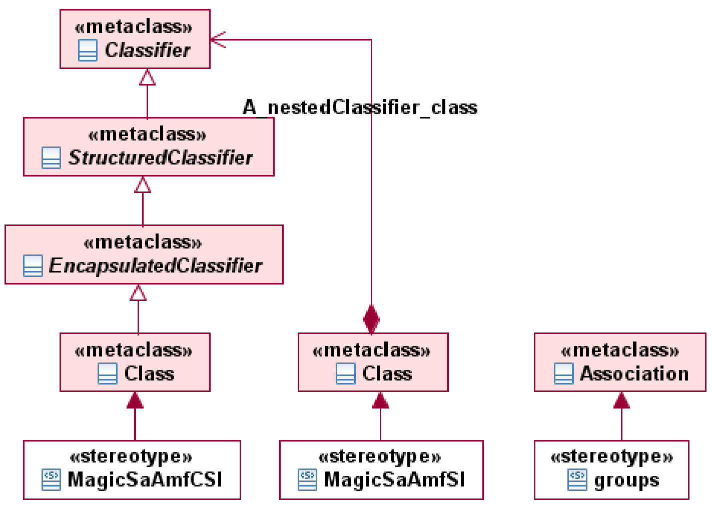


Figure 12 Relationship between SI and CSI

A careful selection (i.e. semantic alignment) of metaclasses for our domain concept related stereotypes allowed us to reuse many associations in the UML metamodel for the aforementioned relationships. Reusing the association from the UML metamodel decreases the complexity of the process of defining the profile while improving the quality of the profile. More specifically, if we consider the related associations of each metaclass as part of its semantic, reusing these associations will implicitly support the semantic alignment and compliance of the domain concepts with respect to the UML metamodel. Each relationship has been stereotyped accordingly and mapped to either Association, AssociationClass, or Dependency.

For example, both <<*MagicSaAmfSI*>> and <<*MagicSaAmfCSI*>> stereotypes are mapped to the UML metaclass Class and, since the metaclass Class inherits indirectly from the metaclass Classifier in the UML metamodel, 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 Figure 12, we defined the stereotype <<groups>> to capture the relationship and map it to metaclass Association. Appendix B summarizes the stereotypes defined for the AMF relationships, their base metaclasses, and the relationship reused from the UML metamodel. More details can be found in [Salehi 2012].

4.2.3 Specifying Constraints

This phase aims at ensuring that the UML stereotyped base metaclasses 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. These constraints can be grouped into two different categories:

Constraints on relationships

For example, the previously defined stereotype <<*groups*>> can be used only between specific AMF entities. However, UML has the capability of using associations between all sorts of UML elements, including the metaclasses Class, Component, and Node. Therefore, without any constraints it would be possible to use the <<*groups*>> relationship to group CSIs 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 metamodel in the context of AMF. For instance, the following constraint restricts the UML metamodel to use the <<*groups*>> stereotype between component and SU:

```
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))
```

Constraints on model elements

Similar to the constraints on relationships, there is another group of constraints that should be taken into account. This group targets UML elements in order to restrict the UML metamodel. For example, based on the AMF domain model, components cannot inherit from other components. However, the UML metamodel allows designers to use inheritance between elements that are mapped to UML metaclass Component. Therefore, another set of constraints was required to restrict the standard UML elements according to what is allowed by AMF. We have defined and specified this set using OCL. The following constraint restricts the inheritance on components:

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

5 Implementation and Case Study

5.1 Environment

We implemented the extension to the UML metamodel in order to model AMF concepts in IBM Rational Software Architect (RSA) [IBM 2014]. RSA is a UML 2.4.1 compliant integrated software development environment which supports UML extension capabilities and which is built on top of the Eclipse platform [Eclipse 2014a]. The combination of RSA and Eclipse Modeling Framework (EMF) [Eclipse 2014b] provides a powerful capability for integrating new domain concepts with UML in a single toolset. By using the visualization and metamodel integration services, RSA integrates different metamodels, allowing them to reference one another. Therefore, it facilitates the model-driven approach for generating, validating, and analyzing models [Leroux 2006].

Compared to other modeling tools, RSA provides its users with a quicker and simpler way of creating UML profiles in order to address domain-specific concerns [Leroux 2006]. In addition, since RSA's internal model representations are based on EMF metamodels, RSA allows users to visualize and integrate models and model elements from different domain formats. Therefore, RSA has a high degree of interoperability with other modelling tools [Leroux 2006].

Our choice of using RSA also lies in the conclusions of the study conducted by Amyot et al. [Amyot 2006]. The authors compare different UML 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. RSA was found one of the most complete tools in its category. Figure 13 shows the implementation of the UACL using RSA. UACL is primarily designed for the automation of the configuration generation process. In previous work, we have used the XML Metadata Interchange (XMI) format of the profile generated by RSA in a model driven configuration generation process. UACL has also been used for the validation of AMF configurations. Configuration design experts can also view, modify and manipulate the AMF configurations using our tool and benefit from RSA's validation engine [Salehi 2010 and Salehi 2012].

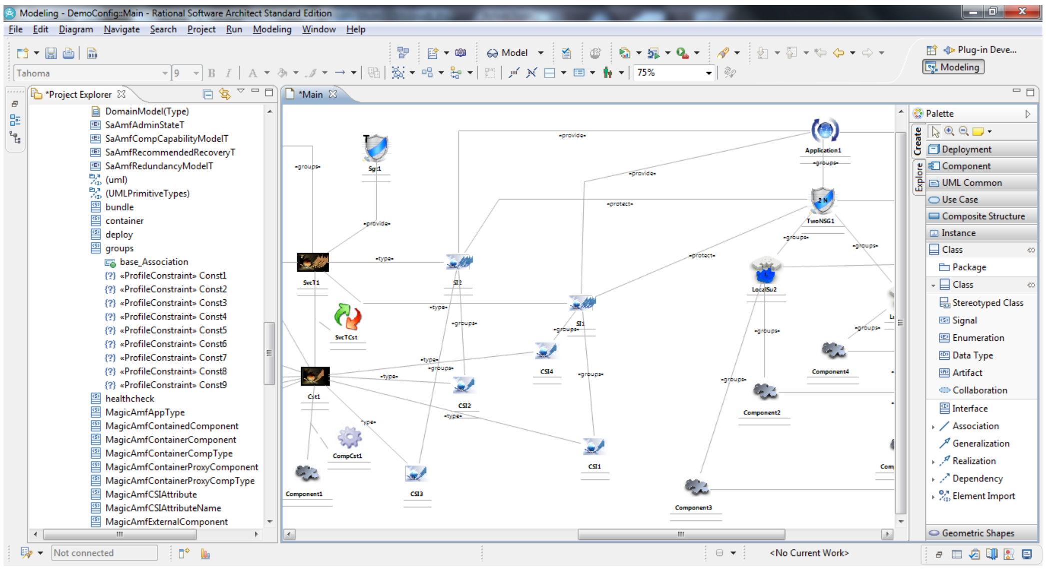


Figure 13 An RSA snapshot of the modeling framework for AMF

Finally, it is worth noting that, based on the AMF standard specification, the configuration needs to be deployed on an SA Forum compliant middleware using an XML file following an Information Model Management (IMM) XML schema. In addition to the tool we have presented in this paper, we have developed a program to map the UACL models into the IMM XML [SAF 2010c] format.

5.2 Case Study

In order to demonstrate its effectiveness, we used our framework to develop a configuration for an online banking system which allows customers to conduct financial transactions using a secure web interface. The features of this system include account transfers, balance inquiries, bill payments, and credit card applications. This case study does not focus on the process of generating the configuration, instead, it presents how a sample AMF configuration can be specified using UACL. In previous work [Salehi 2010], we presented a model-driven solution for the automatic generation of AMF configurations which uses UACL as a core metamodel. The online banking application was designed based on three tier architecture [Sheriff 2006]. Figure 14 presents the architecture of the online banking system. In this paper we focus on the design of an AMF configuration for the business tier of this system which consists of the following subsystems:

- The electronic billing service which allows clients to view and manage their invoices sent by e-mail. It also provides online money transfers from the client's account to a creditor's or vendor's account.
- The authentication service which is responsible for confirming the identity of the clients.
- The fund transfer module which provides four different categories of money transfer services:
    - Transferring money between the different accounts belonging to the same client (e.g. between saving and chequing accounts)
    - Performing money transfers from one client's account to another client's account(s) within the same banking institution
    - Performing money transfers from a client's account to an account held by a different banking institution
    - Transferring funds to the Visa account of a client

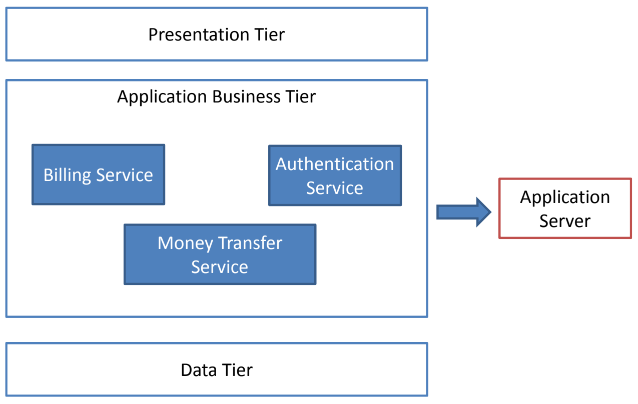


Figure 14 Architecture of the Online Banking System

In this system the user interacts with the system through a web interface and the application layer is supported by an application server. For instance, the application server could be the Red Hat JBoss AS which is an open-source Java EE-based application server [Redhat 2012]. User requests are transferred to the server through Http and the responses are provided accordingly. The availability of the entire set of components in the application layer is managed by the AMF middleware. Moreover, we assume that the APIs required by the AMF middleware are implemented in the software modules and they are capable of supporting all redundancy models.

Figure 15 and Figure 16 represent the entity type view of the FundTransfer part of the configuration. It consists of an SGType grouping the FundTransfer SUType, which provides the FundTransfer service type. The SUType consists of three different component types, namely VisaPayment, MoneyTransfer, and ExternalAccountManager. The first two provide the services for transferring funds to other bank accounts and visa accounts while the ExternalAcountManager is responsible for establishing the connection to the accounts to which the money will be transferred. For this purpose,

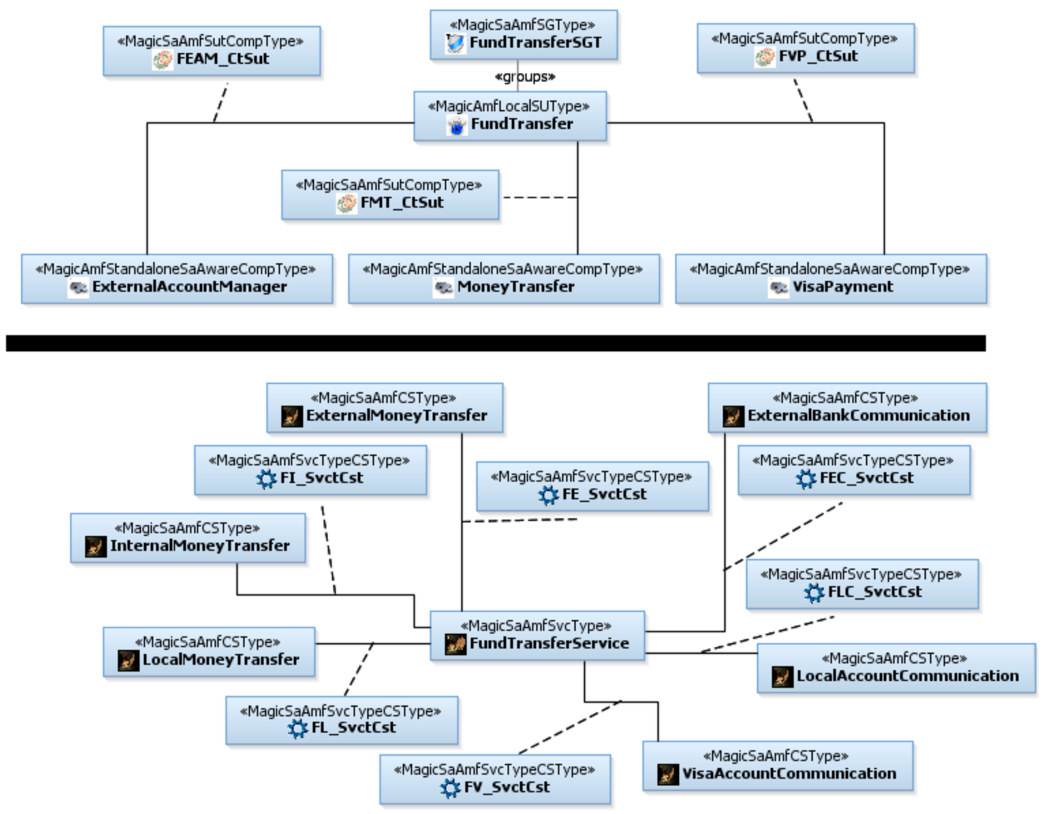

«MagicSaAmfSutCompType» FEAM_CtSut
«MagicSaAmfSGType» FundTransferSGT
«MagicSaAmfSutCompType» FVP_CtSut
«groups»
«MagicAmfLocalSUType» FundTransfer
«MagicSaAmfSutCompType» FMT_CtSut
«MagicAmfStandaloneSaAwareCompType» ExternalAccountManager
«MagicAmfStandaloneSaAwareCompType» MoneyTransfer
«MagicAmfStandaloneSaAwareCompType» VisaPayment
«MagicSaAmfCSType» ExternalMoneyTransfer
«MagicSaAmfCSType» ExternalBankCommunication
«MagicSaAmfSvcTypeCSType» FI_SvctCst
«MagicSaAmfSvcTypeCSType» FE_SvctCst
«MagicSaAmfSvcTypeCSType» FEC_SvctCst
«MagicSaAmfCSType» InternalMoneyTransfer
«MagicSaAmfSvcTypeCSType» FLC_SvctCst
«MagicSaAmfSvcType» FundTransferService
«MagicSaAmfCSType» LocalMoneyTransfer
«MagicSaAmfCSType» LocalAccountCommunication
«MagicSaAmfSvcTypeCSType» FL_SvctCst
«MagicSaAmfSvcTypeCSType» FV_SvctCst
«MagicSaAmfCSType» VisaAccountCommunication

Figure 15 The fund transfer part of the online banking application

MoneyTransfer component type supports LocalMoneyTransfer and ExternalMoneyTransfer CSTypes for transferring the money to other bank accounts held within the same banking institution or in other institutions. The provision of these services depends on the provision of the LocalAccountCommunication and ExternalBankCommunication CSType by ExternalAccountManager. The VisaPayment also provides PayVisaBalance CSType which depends on the provision of the VisaAccountCommunication CSType by ExternalAccountManager. MoneyTransfer component type also provides the InternalMoneyTransfer CSType for transferring funds between the various accounts belonging to the same client (e.g. between savings and chequing accounts) which can be carried out independently.

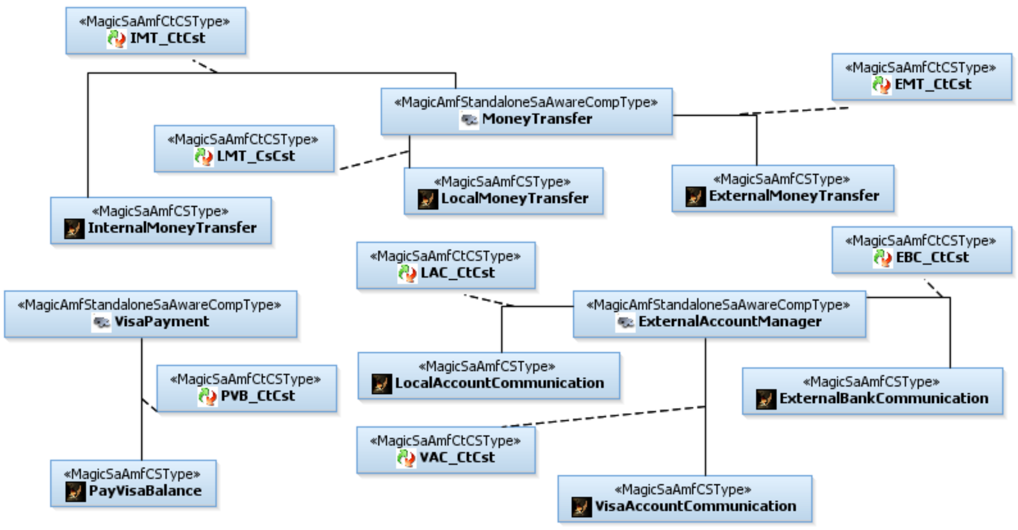

«MagicSaAmfCtCSType» IMT_CtCst
«MagicSaAmfCtCSType» EMT_CtCst
«MagicAmfStandaloneSaAwareCompType» MoneyTransfer
«MagicSaAmfCtCSType» LMT_CsCst
«MagicSaAmfCSType» LocalMoneyTransfer
«MagicSaAmfCSType» ExternalMoneyTransfer
«MagicSaAmfCSType» InternalMoneyTransfer
«MagicSaAmfCtCSType» EBC_CtCst
«MagicSaAmfCtCSType» LAC_CtCst
«MagicAmfStandaloneSaAwareCompType» VisaPayment
«MagicAmfStandaloneSaAwareCompType» ExternalAccountManager
«MagicSaAmfCtCSType» PVB_CtCst
«MagicSaAmfCSType» LocalAccountCommunication
«MagicSaAmfCSType» ExternalBankCommunication
«MagicSaAmfCtCSType» VAC_CtCst
«MagicSaAmfCSType» PayVisaBalance
«MagicSaAmfCSType» VisaAccountCommunication

Figure 16 Component types of the fund transfer part of the online banking application

Figure 17 shows the entity type view of the Billing SUType grouped aggregated by BillingSGT. Billing SUType groups BillManager component type which provides ViewBill and PayBill CSTypes. Provision of the ViewBill CSType depends on the provision of the EPostService by EPostCommunication component type. The provision of the the PayBill CSType is also sponsored by ExternalBankCommunication CSType which is provided by ExternalAccountManager component type (the same component type used in the fund transfer part). Notice that ExternalAccountManager and EPostCommunication component types are aggregated by Billing SUType as well.

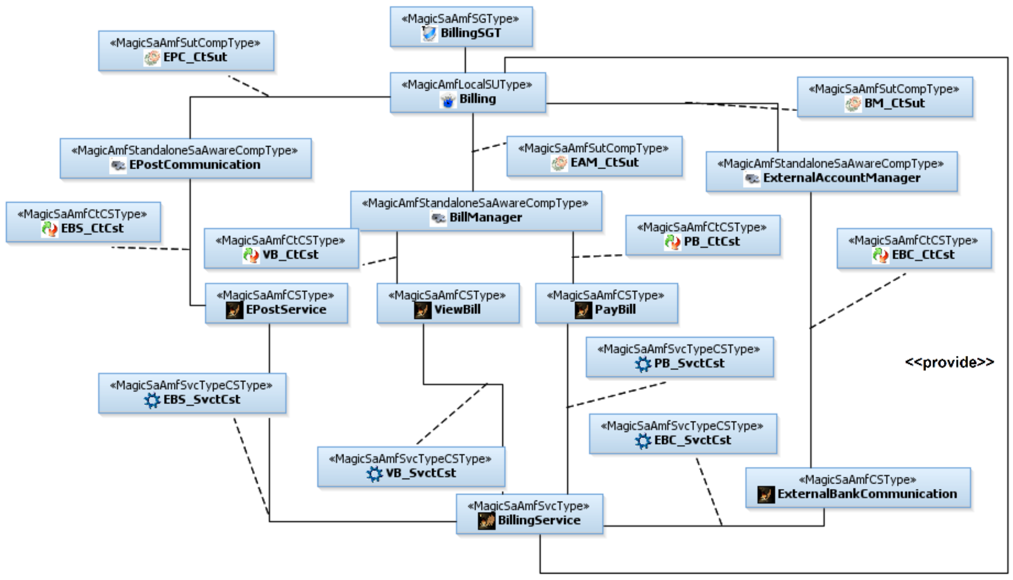

«MagicSaAmfSGType» BillingSGT
«MagicSaAmfSutCompType» EPC_CtSut
«MagicAmfLocalSUType» Billing
«MagicSaAmfSutCompType» BM_CtSut
«MagicAmfStandaloneSaAwareCompType» EPostCommunication
«MagicSaAmfSutCompType» EAM_CtSut
«MagicAmfStandaloneSaAwareCompType» ExternalAccountManager
«MagicSaAmfCtCSType» EBS_CtCst
«MagicAmfStandaloneSaAwareCompType» BillManager
«MagicSaAmfCtCSType» VB_CtCst
«MagicSaAmfCtCSType» PB_CtCst
«MagicSaAmfCtCSType» EBC_CtCst
«MagicSaAmfCSType» EPostService
«MagicSaAmfCSType» ViewBill
«MagicSaAmfCSType» PayBill
«MagicSaAmfSvcTypeCSType» PB_SvctCst
<<provide>>
«MagicSaAmfSvcTypeCSType» EBS_SvctCst
«MagicSaAmfSvcTypeCSType» EBC_SvctCst
«MagicSaAmfSvcTypeCSType» VB_SvctCst
«MagicSaAmfCSType» ExternalBankCommunication
«MagicSaAmfSvcType» BillingService

Figure 17 Entity type view of the billing part of the online banking application

Figure 18 presents the entity type view of the authentication part of the online banking configuration which includes the Security SGType grouping Authentication SUType. Authentication SUType consists of one component type CertifiedAuthentication which provides CertifiedAuthenticationService CSType.

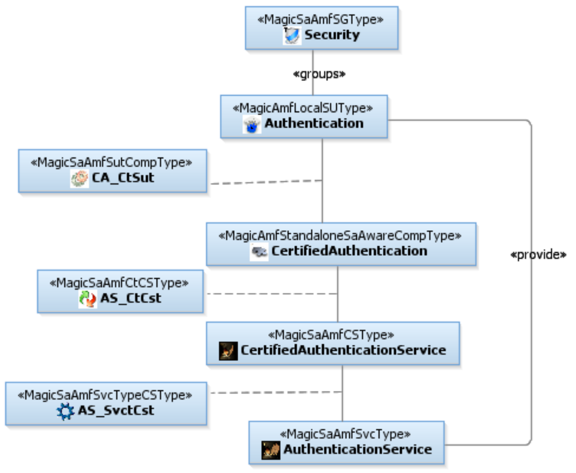

«MagicSaAmfSGType» Security
«groups»
«MagicAmfLocalSUType» Authentication
«MagicSaAmfSutCompType» CA_CtSut
«MagicAmfStandaloneSaAwareCompType» CertifiedAuthentication
«provide»
«MagicSaAmfCtCSType» AS_CtCst
«MagicSaAmfCSType» CertifiedAuthenticationService
«MagicSaAmfSvcTypeCSType» AS_SvctCst
«MagicSaAmfSvcType» AuthenticationService

Figure 18 Entity type view of the security part of the online banking application

Figure 19 represents the entity view of AMF configuration for the online banking application. FundTransfer service group (see Figure 20) which supports N+M redundancy model groups three SUs from FundTransfer SUType, namely FT_Su1, FT_Su2, and FT_Su3. Each SU includes one component from MoneyTransfer, VisaPayment, and ExternalAccountManager component types. For instance, in FT_Su1 Trans_1, Visa_1, and ExtAccMng_1 are instances of the MoneyTransfer, VisaPayment, and ExternalAccountManager, respectively.

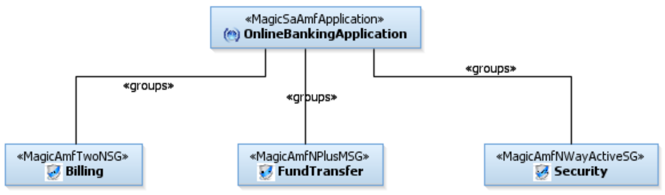

«MagicSaAmfApplication» OnlineBankingApplication
«groups»
«groups»
«groups»
«MagicAmfTwoNSG» Billing
«MagicAmfNPlusMSG» FundTransfer
«MagicAmfNWayActiveSG» Security

Figure 19 Service groups of the online banking application

On the service side, LocalTransServ, VisaServ, and ExternalTransServ SIs are protected by FundTransfer SG. These SIs are instances of the FundTransferService SvcType. However, the CSIs grouped within each one of these SIs are not instances of the same CSTypes. This is possible since the minimum number of the CSIs of each one of the CSTypes in the SIs is zero due to the configuration properties of the FundTransferSevice SvcType and its CSTypes.

It is worth noting that this property has been overridden from the description of the software entities provided by the vendor.

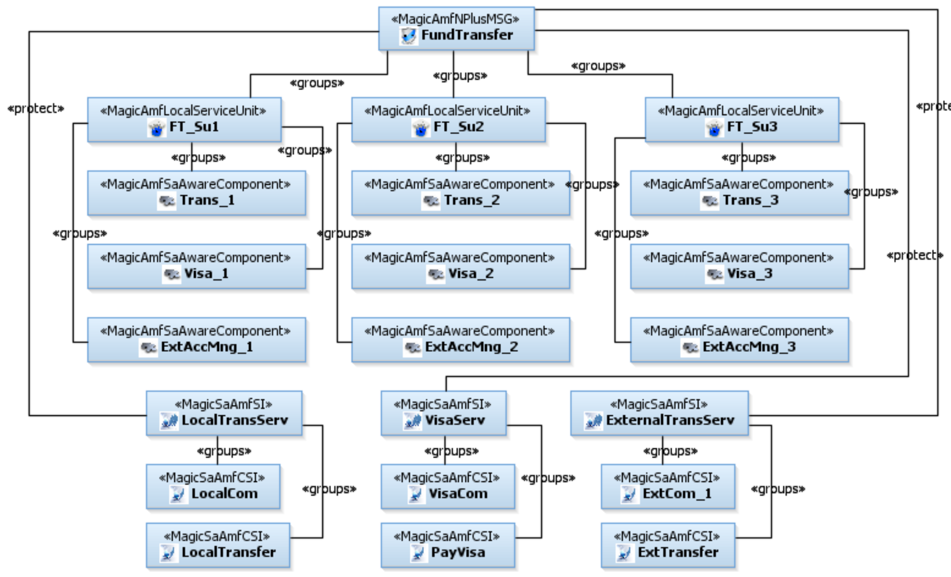

«MagicAmfNPlusMSG» FundTransfer
«protect»
«groups»
«MagicAmfLocalServiceUnit» FT_Su1
«MagicAmfLocalServiceUnit» FT_Su2
«MagicAmfLocalServiceUnit» FT_Su3
«MagicAmfSaAwareComponent» Trans_1
«MagicAmfSaAwareComponent» Trans_2
«MagicAmfSaAwareComponent» Trans_3
«MagicAmfSaAwareComponent» Visa_1
«MagicAmfSaAwareComponent» Visa_2
«MagicAmfSaAwareComponent» Visa_3
«MagicAmfSaAwareComponent» ExtAccMng_1
«MagicAmfSaAwareComponent» ExtAccMng_2
«MagicAmfSaAwareComponent» ExtAccMng_3
«MagicSaAmfSI» LocalTransServ
«MagicSaAmfSI» VisaServ
«MagicSaAmfSI» ExternalTransServ
«MagicSaAmfCSI» LocalCom
«MagicSaAmfCSI» LocalTransfer
«MagicSaAmfCSI» VisaCom
«MagicSaAmfCSI» PayVisa
«MagicSaAmfCSI» ExtCom_1
«MagicSaAmfCSI» ExtTransfer

Figure 20 Entity view of the fund transfer part of the online banking application

More specifically, LocalCom is an instance of the LocalAccountCommunication CSType and LocalTransfer is an instance LocalMoneyTransfer while VisaCom is an instance of the VisaAccountCommunication and PayVisa is an instance of the PayVisaBalance. Moreover, ExtTransfer and ExtCom_1 are instances of the ExternalMoneyTransfer and the ExternalBankCommunication CSTypes, respectively.

Figure 21 shows the entity view of the billing part consists of an SG which protects BillingServ SI from the BillingService SvcType. This SG supports the 2N redundancy model and thus groups two identical SUs (BillingSu1 and BillingSu2) which are instances of the Billing SUType. Each SU includes one component of EPostCommunication, BillManager, and ExternalAccountManager component types. On the service side, the BillingServ SI groups EPost, BillView, BillPayment, and ExtCom_2 CSIs which are instances of the EPostService, ViewBill, PayBill, and ExternalBankCommunication CSTypes, respectively.

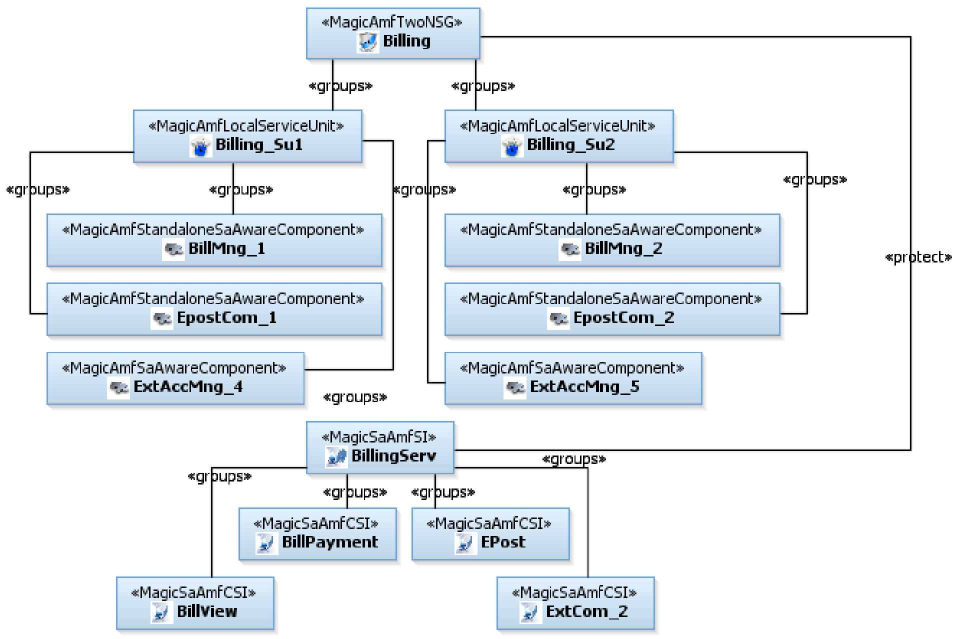

«MagicAmfTwoNSG» Billing
«groups»
«groups»
«MagicAmfLocalServiceUnit» Billing_Su1
«MagicAmfLocalServiceUnit» Billing_Su2
«groups»
«groups»
«groups»
«groups»
«groups»
«MagicAmfStandaloneSaAwareComponent» BillMng_1
«MagicAmfStandaloneSaAwareComponent» BillMng_2
«protect»
«MagicAmfStandaloneSaAwareComponent» EpostCom_1
«MagicAmfStandaloneSaAwareComponent» EpostCom_2
«MagicAmfSaAwareComponent» ExtAccMng_4
«MagicAmfSaAwareComponent» ExtAccMng_5
«groups»
«MagicSaAmfSI» BillingServ
«groups»
«groups»
«groups»
«groups»
«MagicSaAmfCSI» BillPayment
«MagicSaAmfCSI» EPost
«MagicSaAmfCSI» BillView
«MagicSaAmfCSI» ExtCom_2

Figure 21 Entity view of the billing part of the online banking application

Figure 22 presents the entity view of the security module of the online banking system which includes Security SG with the N-Way-Active redundancy model. This redundancy model aims at balancing the load of the authentication service on the protected SIs. Security SG includes two SUs, namely Sec_Su1 and Sec_Su2 from Authentication SUType. Each of these SUs aggregates one component of CertifiedAuthentication component type. On the service side, Security SG protects three identical SIs which are the instances of the AuthenticationService SvcType and each one has one CSI of the CertifiedAuthenticationService CSType.

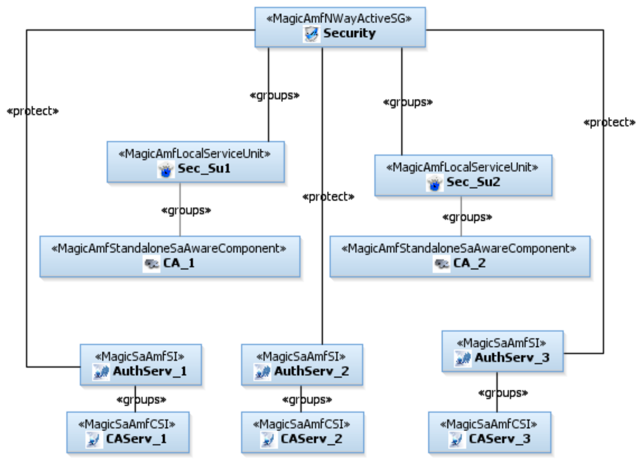
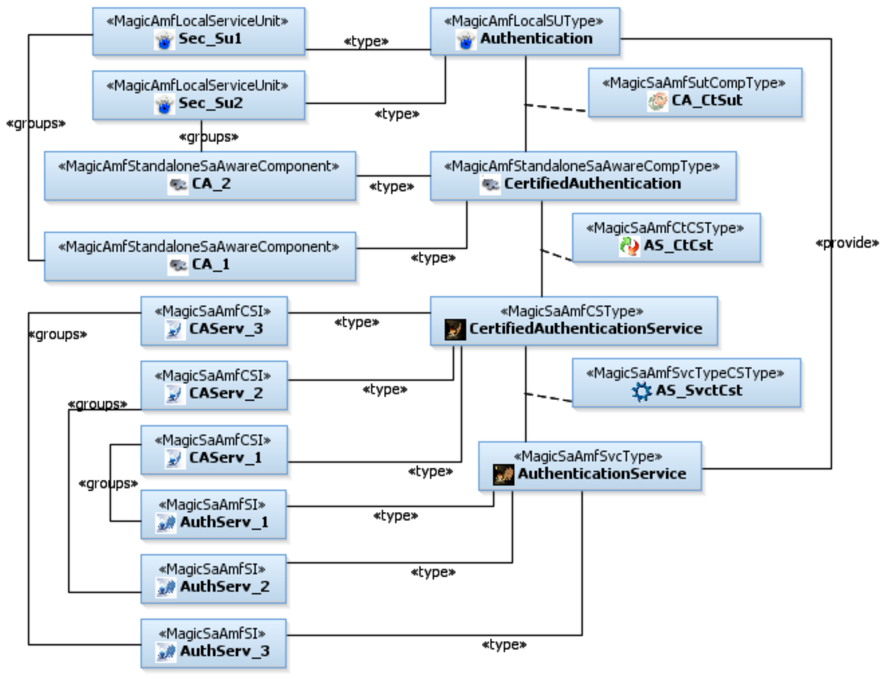

«MagicAmfNWayActiveSG» Security
«groups»
«groups»
«protect»
«protect»
«MagicAmfLocalServiceUnit» Sec_Su1
«MagicAmfLocalServiceUnit» Sec_Su2
«protect»
«groups»
«groups»
«MagicAmfStandaloneSaAwareComponent» CA_1
«MagicAmfStandaloneSaAwareComponent» CA_2
«MagicSaAmfSI» AuthServ_1
«MagicSaAmfSI» AuthServ_2
«MagicSaAmfSI» AuthServ_3
«groups»
«groups»
«groups»
«MagicSaAmfCSI» CAServ_1
«MagicSaAmfCSI» CAServ_2
«MagicSaAmfCSI» CAServ_3

Figure 22 Entity view of the security part of the online banking application

It is worth noting that, since the entity and entity type views are presented separately in this paper, the relationship between each entity and its type entity is not visible in the diagrams. For instance, Figure 23 presents the type relationship between the entities and entity types of the authentication module of the online banking system.



«MagicAmfLocalServiceUnit» Sec_Su1
«MagicAmfLocalSUType» Authentication
«MagicAmfLocalServiceUnit» Sec_Su2
«MagicSaAmfSutCompType» CA_CtSut
«MagicAmfStandaloneSaAwareComponent» CA_2
«MagicAmfStandaloneSaAwareCompType» CertifiedAuthentication
«MagicSaAmfCtCSType» AS_CtCst
«MagicAmfStandaloneSaAwareComponent» CA_1
«MagicSaAmfCSI» CAServ_3
«MagicSaAmfCSType» CertifiedAuthenticationService
«MagicSaAmfCSI» CAServ_2
«MagicSaAmfSvcTypeCSType» AS_SvctCst
«MagicSaAmfCSI» CAServ_1
«MagicSaAmfSvcType» AuthenticationService
«MagicSaAmfSI» AuthServ_1
«MagicSaAmfSI» AuthServ_2
«MagicSaAmfSI» AuthServ_3
«type»
«groups»
«provide»

Figure 23 The relationship between entity and entity type view of the security parts

The configuration is deployed in a cluster called OnlineBanking_Cluster which consists of three AMF nodes (Node1, Node2, and Node3). Node1 is hosting Sec_Su1 and FT_Su1 while Node3 is hosting FT_Su3 and Billing_Su1. Sec_Su2, FT_Su2, and Billing_Su2 are hosted by Node3. Figure 24 shows the deployment view of the AMF configuration designed for online banking application using UACL.

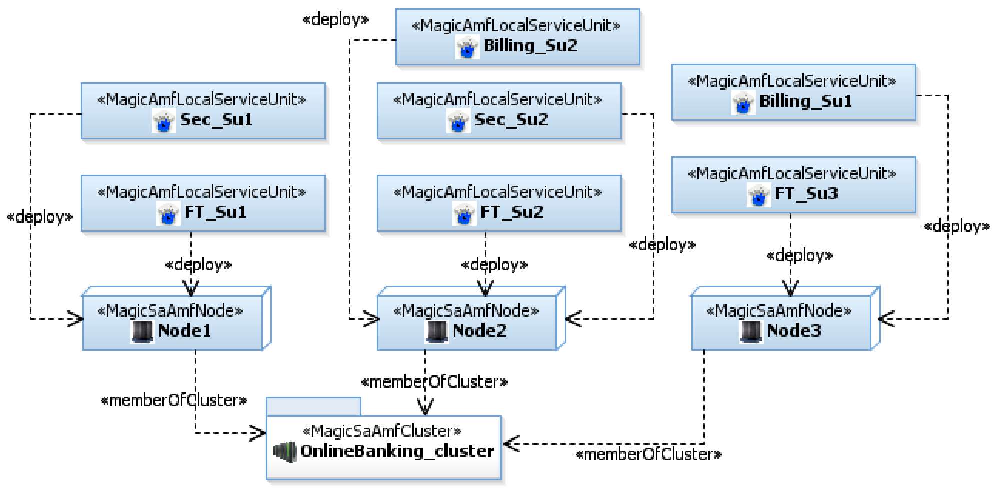

«deploy»
«MagicAmfLocalServiceUnit» Billing_Su2
«MagicAmfLocalServiceUnit» Sec_Su1
«MagicAmfLocalServiceUnit» Sec_Su2
«MagicAmfLocalServiceUnit» Billing_Su1
«MagicAmfLocalServiceUnit» FT_Su1
«MagicAmfLocalServiceUnit» FT_Su2
«MagicAmfLocalServiceUnit» FT_Su3
«deploy»
«deploy»
«deploy»
«deploy»
«deploy»
«deploy»
«MagicSaAmfNode» Node1
«MagicSaAmfNode» Node2
«MagicSaAmfNode» Node3
«memberOfCluster»
«memberOfCluster»
«MagicSaAmfCluster» OnlineBanking_cluster
«memberOfCluster»

Figure 24 Deployment view of the AMF configuration for online banking application expressed in UACL

Finally, it is worth noting that RSA's live mode validation of OCL constraints ensures the validity of the designed configuration according to the standard specification. Live validation mode is not practical, however, for complex constraints involving various modeling elements, as it decreases the tool's level of performance. For this group of constraints, batch mode validation is the most preferable option. This results in the need to validate the entire configuration using RSA after having completed the design. In the occurrence of an error, RSA locates the error so that the designer can detect the cause and fix it.

6 Challenges

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 MOF-based language, we avoided a MOF-based solution as it would suffer from a lack of tool support. In general, the advantages of an UML profile seem to far outweigh its drawbacks [Fuentes 2004]. The second issue was in deciding 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 [OMG 2011], SPT [OMG 2005], and QoS&FT [OMG 2008], as well as the non-standardized profiles reported in the literature, such as the DAM [Bernardi 2011] profile and the profile presented by Szatmári et al. in [Szatmári

2008]. 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 metamodel. 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 metamodel and not go through the DAM profile.
2. The implementation of the existing profiles. One of the goals of this work was 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 needed 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 metamodel instead of reusing another profile and adapting it to AMF.

The AMF specification defines the run-time behaviour of the middleware with respect to the management of the AMF configurations. At the domain modeling stage, one of the most challenging issues was to capture only the configuration time aspects. In other words, we extracted the configuration time aspects of the configuration by analyzing the run-time behaviour of the AMF. It was not straightforward to extract the domain model from the large standard document, as the domain model requires the isolation of the configuration time characteristics from the run-time characteristics of the AMF. More specifically, some of the concepts defined at run-time are based on other related configuration time constraints in order to ensure that the configured application will provide and protect the service independently from a particular AMF service implementation. This also increases the complexity of the domain model, both in the design of the class diagram and in the process of specifying the OCL constraints. For instance, SA Forum standards support the notion of proxied components –the components which need proxy components to interact with the AMF middleware. However, the links between a proxy component and its proxied components are established only at run-time when an AMF service implementation selects and assigns a particular proxy component to a particular proxied component. On the other hand, a configuration time relationship between a proxy and a proxied component is specified through the proxyCSI. This is a particular CSI through which a proxy component is assigned the task of "proxying" a particular proxied component. The concept of proxyCSI was captured in our model through the association end magicSaAmfCompProxyCSI of the association between MagicLocalProxiedComponent and MagicSaCSI classes as shown in Figure 25. In order to configure a certain proxy component for its proxied components, one can only use the proxyCSI. Therefore, the well-formedness of the configuration can be expressed and checked at configuration time to the extent allowed by the proxyCSI configuration attribute. At configuration time, we have to ensure that there is at least one proxy component capable of being the proxy component for the proxied component. This constraint translates to the existence of a proxy component that supports a component service type (CSType) to which the proxyCSI of the proxied component in question belongs. The OCL expression formalizing this domain constraint is as follows:

```
contex MagicAmfLocalProxiedComponent
inv:
(MagicSaAmfSI.allInstances->
  select(s : MagicSaAmfSI | s.magicSaAmfSiGroups->
    includes(self.magicSaAmfCompProxyCsi))->
      forAll(s2 :MagicSaAmfSI|
       s2.magicSaAmfSiProtectedbySG.magicAmfSGGroups->
        forAll(su : MagicSaAmfSU |
         su.oclIsTypeOf(MagicAmfLocalServiceUnit)
          and
           su.magicAmfLocalServiceUnitGroups->
            select(c|c.oclIsTypeOf(MagicAmfProxyComponent)->
             iterate(v:MagicAmfProxyComponent , a = Set{} |
              v.oclIsTypeOf(MagicAmfProxyComponent))
               implies
               (a-> union(v.MagicSaAmfCompCsType.
                  magicSafSupportedCsType))-> includes
            (self.magicSaAmfCompProxyCsi.magicSaAmfCsType))
```

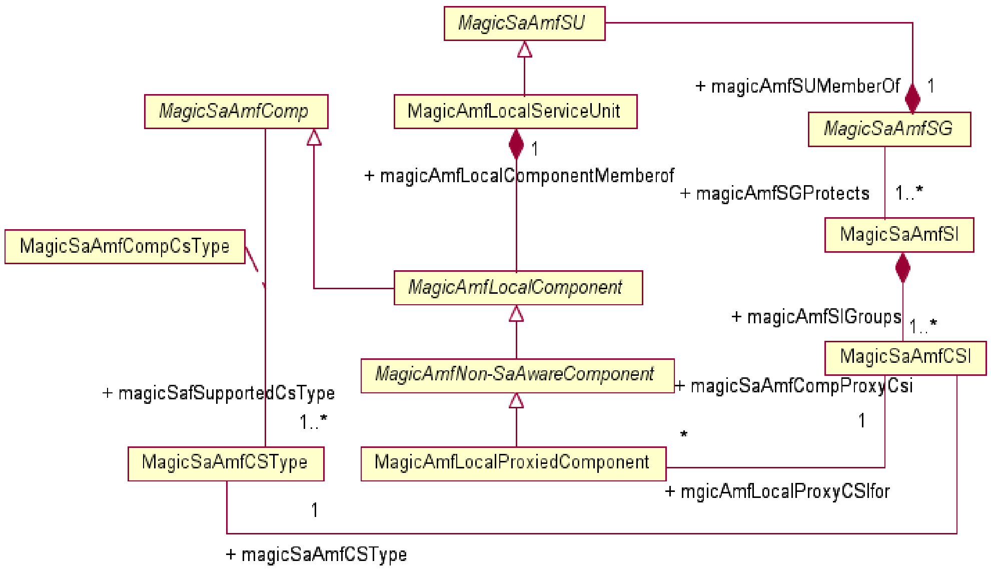


Figure 25 Proxy-proxied Component Relationship

Another challenge that we encountered was the identification of UML metaclasses for mapping purposes. More precisely, we had to identify the most appropriate UML metaclasses 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.

In addition, a complementary and important aspect needs to be taken into consideration: the tool support. We chose RSA because of its features. However, our experience with RSA also revealed some of its 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 are not compliant with the standard OCL specification and therefore, standard OCL constraints cannot directly be implemented in RSA. For instance, in the context of an entity, if we want to verify the stereotype as being set to <<*MagicSaAmfLocalSU*>>, we will need the following OCL constraint:

```
self. isStereotypeApplied
(MAGICAMFProfile::MagicSaAmfSU.oclAsType(uml::Stereotype))
```

As observed in the above example, additional type casting commands are required in order for the constraint to perform properly. Considering the fact that almost all of the constraints in the UML profiles deal with stereotypes, this drawback has a great impact on the readability of the OCL constraints and therefore, the maintainability of the tool.

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– so as 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.

One of the main limitations of UACL comes from its inherent complexity rooted in the intricate concepts and their relationships defined in the standard specifications. This complexity does pose limitations on the manual manipulation of the configurations and requires designers to possess a strong understanding of the language and its elements. This complexity can also hinder the maintainability of the profile. Since the main objective of UACL is to provide a modeling framework for the automatic generation of configurations, the complexity of the language has minimal consequences on the effectiveness of the language. This complexity does not affect the automatic configuration generation process

The other limitation comes from the lack of formal processes to validate the language in order to ensure its compliance with the standard specifications. We have invested a great deal of effort in defining our profile by refining and reusing a generic process discussed in [Selic 2007]. In addition, our work has undergone an intensive and effective review process with the domain expert and with a team of experienced software designers. However, the lack of a well-defined evaluation mechanism and metrics for (formally) evaluating our UML profiles seems to be a limitation that needs to be addressed in the future. The applicability and usefulness of UACL will be evaluated empirically over time. This will help us improve the profile, enhance the guidelines for defining a UML profile, and perhaps design an evaluation framework.

7 Conclusion

An AMF configuration is the artefact used by an implementation of the AMF middleware service 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 metamodel. 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 metamodel and the definition of constraints.

The experience, as discussed in Section 4, has shown that the most critical aspects of this process were 1) the domain analysis, like dropping the run-time attributes and classes from the domain model, which led to difficulties in specifying the related configuration concepts, such as the domain constraints necessary for the definition of AMF configurations, 2) the identification of metaclasses for mapping purposes, and 3) having adequate tool support. Due to the existing relationships at the level of the UML metaclasses, 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 [Salehi 2010].

Acknowledgments

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

References

[Aagedal 2005]	J. Aagedal, J. Bezivin, and P. Linington, "Model-driven development," 2005. in: Malenfant, J. and Ostvold, Bjarte.M., eds. ECOOP 2004 Workshop Reader. LNCS, 3344. Springer-Verlag, pp. 148-157.
[Amyot 2006]	D. Amyot and J. Roy, Evaluation of Development Tools for Domain-Specific Modeling Languages. In: 5th Int. Workshop on System Analysis and Modeling, LNCS v. 4320 (2006).
[AUTOSAR 2006]	AUTOSAR GbR, UML Profile for AUTOSAR Specification, Version 1.0.1. 2006, URL: http://www.autosar.org.
[Belloni 2006]	E. Belloni, and C. Marcos, "MAM-UML: An UML Profile for the Modeling of Mobile-Agent Applications," in Proc. of the 24th International Conference of the Chilean Computer Science Society, 2004, pp.3-13.
[Bernardi 2011]	S. Bernardi, J. Merseguer, and D. Petriu, "A dependability profile within MARTE" Software and System Modeling 10(3): 313-336 (2011)
[Corosync 2015]	Corosync Cluster Engine, http://corosync.github.io/corosync/, accessed July 2015

[Eclipse 2014a] Eclipse Foundation, 2014, URL: http://www.eclipse.org/

[Eclipse 2014b] Eclipse Foundation, Eclipse Modeling Framework (EMF), 2014, URL: http://www.eclipse.org/modeling/emf/

[Fuentes 2004] L. Fuentes-Fernández, and A. Vallecillo-Moreno, An introduction to UML profiles. The European Journal for the Informatics Professional, Vol. 5, No. 2, 2004.

[Gherbi 2009] A. Gherbi, P. Salehi, F. Khendek and A. Hamou-Lhadj "Capturing and Formalizing SAF Availability Management Framework Configuration Requirements," in Proc. of the First International Workshop on Domain Engineering (DE@CAiSE'09) 2009.

[IBM 2014] Rational Software Architect, IBM Rational Software 2013, www-03.ibm.com/software/products/en/ratisoftarchfami

[ITU 2007] International Telecommunication Union (ITU), "Z.119 : Guidelines for UML profile design," 2007, URL: https://www.itu.int/rec/T-REC-Z.119-200702-I/en

[Kanso 2008] A. Kanso, M. Toeroe, F. Khendek, and A. Hamou-Lhadj, “Automatic Generation of AMF Compliant Configurations,” in Proc. of the 5th International Service Availability Symposium, Tokyo, Japan, 2008 pp. 155-170.

[Kanso 2009] A. Kanso, M. Toeroe, A. Hamou-Lhadj, and F. Khendek, “Generating AMF Configurations from Software Vendor Constraints and User Requirements,” in Proc. of the Forth International Conference on Availability, Reliability and Security, Fukuoka, Japan, 2009, pp. 454-461.

[Kövi 2007] A. Kövi, et al., UML profile and design patterns library. (Preliminary version), Aalborg University, Aalborg, Denmark, IST-FP6-STREP-26979 / HIDENETS, 2007.

[Lagarde 2007] F. Lagarde, et al., "Improving UML profile design practices by leveraging conceptual domain models," in Proc. of the 22nd IEEE/ACM International Conference on Automated Software Engineering, Atlanta, USA, 2007, pp. 445-448.

[Lagarde 2008] F. Lagarde, et al., "Leveraging patterns on domain models to improve uml profile definition," in Proc of the Theory and practice of software, 11th international conference on Fundamental approaches to software engineering, Budapest, Hungary, 2008, pp. 116-130.

[Leroux 2006] D. Leroux, M. Nally, K. Hussey "Rational Software Architect: A tool for domain-specific modeling", IBM System Journal, 2006.

[MontaVista 2015] MontaVista, http://www.mvista.com/company.html, accessed July 2015.

[OMG 2002] Object Management Group, Common Object Request Broker Architecture: Core Specification, Version 3.0.2, formal/02-12-02, 2002, URL: http://www.omg.org/cgi-bin/doc?formal/02-12-02.

[OMG 2005] Object Management Group, UML Profile for Schedulability, Performance, and Time Specification, Version 1.1, formal/2005-01-02, 2005, URL: http://www.omg.org/spec/SPTP/1.1/.

[OMG 2007a] Object Management Group, XML Metadata Interchange (XMI) Specification, Version 2.1.1, formal/2007-12-02, 2007, URL: http://www.omg.org/spec/XMI/2.1.1/.

[OMG 2007b] Object Management Group, Unified Modeling Language - Superstructure Version 2.1.1 formal/2007-02-03, 2007, URL: http://www.omg.org/technology/documents/formal/uml.htm.

[OMG 2008] Object Management Group, UML Profile for Modelling Quality of Service and Fault Tolerance Characteristics and Mechanisms Specification, formal/2008-04-05, 2008, URL: http://www.omg.org/spec/QFTP/1.1/.

[OMG 2010] Object Management Group, Fault Tolerant CORBA, formal/2010-05-07, 2010, URL: http://www.omg.org/spec/FT/1.0.

[OMG 2011] Object Management Group, A UML Profile for MARTE Specification, Version 1.1, formal/2011-06-02, URL: http://www.omg.org/spec/MARTE/1.1/PDF/

[OMG 2012a] Object Management Group, Lightweight Fault Tolerance for Distributed RT Systems (LWFT), Version 1.0, formal/2012-03-02, URL: http://www.omg.org/spec/LWFT/1.0/PDF

[OMG 2012b] Object Management Group, SysML Specification, Version 1.2 formal/10-06-02, 2010, URL: http://www.sysml.org/specs.htm.

[OMG 2012c] Object Management Group, Unified Modeling Language (UML) Superstructure, Version 2.4.1, formal/2011-08-06, 2012, URL: http://www.omg.org/spec/UML/2.4.1/Superstructure.

[OMG 2014a] OMG, Object Constraint Language, Version 2.2 - http://omg.org/spec/OCL/2.4/PDF/

[OMG 2014b] Object Management Group, URL: http://www.omg.org

[OpenAIS 2015] OpenAIS, OpenAIS Overview, http://linux.die.net/man/8/openais_overview, accessed July 2015.

[OpenHPI 2015] OpenHPI, http://openhpi.org/, accessed July 2015.

[OpenSAF 2015] OpenSAF Foundation, http://www.opensaf.org/, accessed July 2015.

[Redhat 2012] JBoss Application Server, URL: http://www.jboss.org/

[SAF 2014] Service Availability Forum™, URL: http://www.saforum.org

[SAF 2010a] Service Availability Forum™, Overview SAI-Overview-B.05.03 at: http://www.saforum.org/link/linkshow.asp?link_id=222259&assn_id=16627

[SAF 2010b] Service Availability Forum™, Hardware Platform Interface SAI HPI-B.03.02 at: http://www.saforum.org/link/linkshow.asp?link_id=222259&assn_id=16627

[SAF 2010c] Service Availability Forum™, Application Interface Specification. Availability Management Framework SAI-AIS-AMF-B.04.01

[SAF 2010d] Service Availability Forum, Application Interface Specification. Software Management Framework SAI-AIS-SMF-A.01.01.

[Salehi 2009] P. Salehi, et al. "Checking Service Instance Protection for AMF Configurations," in Proc. of the Third IEEE International Conference o Secure Software Integration and Reliability Improvement, Shanghai, China, 2009, pp. 269 - 274.

[Salehi 2010] P. Salehi, P. Colombo, A. Hamou-Lhadj, and F. Khendek, "A Model Driven Approach for AMF Configuration Generation," in Proc. of 6th Workshop on System Analysis and Modelling, Oslo, Norway, 2010, pp. 124-143.

[Salehi 2012] P. Salehi,"A Model Based Framework for Service Availability Management", Doctoral dissertation, Concordia University, 2012.

[Selic 2003] B. Selic, "The pragmatics of model-driven development," in IEEE software, 2003. 20(5), pp. 19-25.

[Selic 2007] B. Selic, "A systematic approach to domain-specific language design using UML," in Proc. of the 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07), Santorini Island, Greece, 2007, pp. 2-9.

[Sheriff 2006] Paul D Sheriff, *Fundamentals of N-Tier Architecture*, [S.I.], PSDA, 2006.

[Szatmári 2008] Z. Szatmári, A. Kövi, and M. Reitenspiess, "Applying MDA approach for the SA forum platform," in Proc of the 2nd workshop on Middleware-application interaction, Oslo, Norway, 2008, pp. 19-24.

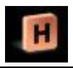
[Toeroe 2012] M. Toeroe, and F. Tam, (Eds.), "Service availability: principles and practice," John Wiley & Sons, 2012.

[Turenne 2014a] M. Turenne, A. Kanso, A. Gherbi, S. Razzook, "A tool chain for generating the description files of highly available software," in Proc of the 29th International Conference on Automated Software Engineering, Vasteras Sweden, 2014,pp. 867-870.

[Turenne 2014b] M. Turenne, A. Kanso, A. Gherbi, R. Barrett "Automatic Generation of Description Files for Highly Available Services," in Proc of the 6th International Workshop on Software Engineering for Resilient Systems, Budapest, Hungary, 2014, pp. 40-54.

Appendix A

Table 2 The summary of the stereotypes defined for AMF entities and entity types

Stereotype	Generalization	Notation
<<MagicSaAmfCompGlobalAttributes>>	**metaclass Class**	
<<SaAmfCompBaseType>>	**metaclass Class**	
<<MagicSaAmfCompType >>	<<SaAmfCompBaseType>>	
<<MagicAmfSaAwareCompType>>	*<<MagicSaAmfCompType>>*	
<<MagicAmfStandaloneSaAwareCompType >>	<<MagicAmfSaAwareCompType>>	
<<MagicAmfProxyCompType>>	<<MagicAmfStandaloneSaAwareCompType>>	
<<MagicAmfContainerCompType>>	<<MagicAmfStandaloneSaAwareCompType>>	
<<MagicAmfContainer-ProxyCompType>>	<<MagicAmfProxyCompType>> <<MagicAmfContainerCompType>>	
<<MagicAmfProxiedCompType>>	<< MagicSaAmfCompType>>	
<<MagicAmfNon-ProxiedNon-SaAwareCompType>>	<< MagicSaAmfCompType>>	
<<MagicSaAmfHealthcheckType>>	**metaclass Class**	
<<SaAmfSUBaseType>>	**metaclass Class**	
<<MagicSaAmfSUType>>	<<SaAmfSUBaseType>>	
<<MagicAmfLocalSUType>>	*<<MagicSaAmfSUType>>*	
<<MagicAmfExternalSUType>>	*<<MagicSaAmfSUType>>*	

<<SaAmfSGBaseType>>	**metaclass Class**	
<<MagicSaAmfSGType>>	<<SaAmfSGBaseType>>	
<<SaAmfAppBaseType>>	**metaclass Class**	
<<MagicAmfAppType >>	<<SaAmfAppBaseType>>	
<<SaAmfCSBaseType>>	**metaclass Class**	
<<MagicSaAmfCSType>>	<<SaAmfCSBaseType>>	
<<SaAmfSvcBaseType>>	**metaclass Class**	
<<MagicSaAmfSvcType>>	<<SaAmfSvcBaseType>>	
<<MagicSaAmfComp>>	**metaclass Component**	
<<MagicAmfLocalComponent>>	*<<MagicSaAmfComp>>*	
<<MagicAmfExternalComponent>>	*<<MagicSaAmfComp>>*	
<<MagicAmfSaAwareComponent>>	*<<MagicAmfLocalComponent>>*	
<<MagicAmfNon-SaAwareComponent>>	*<<MagicAmfLocalComponent>>*	
<<MagicAmfStandaloneSaAwareComponent>>	*<<MagicAmfSaAwareComponent>>*	
<<MagicAmfContainedComponent>>	*<<MagicAmfSaAwareComponent>>*	
<<MagicAmfLocalProxiedComponent>>	*<<MagicAmfNon-SaAwareComponent>>*	
<<MagicAmfNon-ProxiedNon-SaAwareComponent>>	*<<MagicAmfNon-SaAwareComponent>>*	
<<MagicAmfContainerComponent>>	<<MagicAmfStandaloneSaAwareComponent>>	
<<MagicAmfProxyComponent>>	<<MagicAmfStandaloneSaAwareComponent>>	

<<MagicAmfContainer-ProxyComponent>>	<<MagicAmfContainerComponent>> <<MagicAmfProxyComponent>>	
<<MagicSaAmfHealthcheck>>	**metaclass Class**	H
<<MagicSaAmfSU>>	**metaclass Component**	
<<MagicAmfLocalServiceUnit>>	*<<MagicSaAmfSU>>*	L
<<MagicAmfExternalServiceUnit>>	*<<MagicSaAmfSU>>*	E
<<MagicSaAmfSG>>	**metaclass Component**	
<<MagicAmfTwoNSG>>	*<<MagicSaAmfSG>>*	2 N
<<MagicAmfNPlusMSG>>	*<<MagicSaAmfSG>>*	N + M
<<MagicAmfNWaySG>>	*<<MagicSaAmfSG>>*	N W
<<MagicAmfNWayActiveSG>>	*<<MagicSaAmfSG>>*	N W A
<<MagicAmfNoRedundancySG>>	*<<MagicSaAmfSG>>*	N R
<<MagicSaAmfApplication>>	**metaclass Component**	
<<MagicSaAmfCSI>>	**metaclass Class**	
<<MagicSaAmfSI>>	**metaclass Class**	
<<MagicAmfCSIAttributeName>>	**metaclass Class**	
<<MagicSaAmfNode>>	**metaclass Node**	
<<MagicSaAmfNodeGroup>>	**metaclass Package**	
<<MagicSaAmfCluster>>	**metaclass Package**	

<<MagicSaSmfSwBundle>>	**metaclass Class**	📦

Appendix B

Table 3 Summary of Stereotypes Related to the Relationships between Domain Concepts in AMF

Stereotype	**UML metaclass**	**Reused relationship from UML metamodel**
<<groups>>	**metaclass Association**	nestedClassifier relationship between Class and Classifier packagedElement relationship between Componnet and Packageable Element
<<protect>>	**metaclass Association**	nestedClassifier relationship between Class and Classifier
<<provide>>	**metaclass Association**	nestedClassifier relationship between Class and Classifier
<<type>>	**metaclass Association**	superClass relationship between Componnet and Class Reflective superClass relationship on Class
<<membernode>>	**metaclass Dependency**	packagedElement relationship between Packageable Element and Package
<<deploy>>	**metaclass Dependency**	packagedElement relationship between Packageable Element and Package
<<MagicSaAmfSutCompType>>	**metaclass AssociationClass**	nestedClassifier relationship between Class and Classifier
<<MagicSaAmfSvcTypeCSType>>	**metaclass AssociationClass**	packagedElement relationship between Componnet and Packageable Element.
<<MagicSaAmfCtCSType>>	**metaclass AssociationClass**	nestedClassifier relationship between Class and Classifier
<<MagicSaAmfCompCsType>>	**metaclass AssociationClass**	nestedClassifier relationship between Class and Classifier
<<MagicSaAmfSIDependency>>	**AssociationClass**	nestedClassifier relationship between Class and Classifier inherited by AssociationClass