Capturing and Formalizing SAF Availability Management Framework Configuration Requirements

Completed Research

Abstract. The Service Availability Forum (SAF) defines a set of middleware services to support and enable high availability in a standardized manner. The Availability Management Framework (AMF) is the service in charge of managing the high availability of the services provided by an application under its control. In order to do so, the AMF service requires a configuration of the application, referred to as an AMF configuration. We are currently defining a UML profile for the modeling and analysis of AMF configurations. The configuration model and the runtime behavior of an AMF service implementation as a middleware are both defined in the AMF specification. It is not straightforward to extract from the large standard document the domain model, which requires the isolation of the configuration time characteristics from the runtime characteristics of the AMF service. In this paper, we report on our experience in designing a domain model for AMF configurations; we discuss some of the challenges we encountered during this process. We also discuss how this domain model is used for the validation of AMF configurations.

Keywords: High-Availability, Service Availability Forum, Availability Management Framework, Domain model, UML, OCL

1 Introduction

Availability is an important characteristic of dependable systems along with reliability and security [1]. It is often achieved using proprietary solutions based on redundancy and clustering in order to eliminate single point of failures. High Availability (HA) is a more stringent requirement where the system should be up and running in the order of five nines and higher (99.999%) [2]. Such a high level of availability is expected from service providers in telecommunication and banking, for instance.

Service Availability Forum (SAF) [3], is a consortium of telecommunication and computer companies, that defines and supports HA standard specifications. In particular, the SAF Application Interface Specification (AIS) [4] includes the specification of the Availability Management Framework (AMF) service [5], which is the SAF middleware service responsible of managing service availability through the coordination of redundant resources.

To manage the availability of the services delivered by an application under its control, an AMF service implementation requires a configuration, which describes the different resources in use, their capabilities and limitations, their organization, relations, the services to be provided, and their protection. The design and the upgrade of such configurations have to be done very carefully in order to meet the high availability requirement. We are undertaking a research project aiming at devising methods and prototype tools for the design and analysis of AMF configurations as well as their upgrade campaigns. In order to achieve this formally, we started with the definition of a profile of the Unified Modeling Language (UML) [16] for AMF configurations.

The first step in the process of defining a UML profile is the elaboration of a domain model [6]. This model captures the main concepts of the domain, their relationships and constraints. Our main input for establishing the domain model is the AMF standard specification [5]. The B.03.01 version of this specification is a 395-pages document, which is mostly informal. This document defines informally an AMF configuration as well as the runtime behavior of an AMF service implementation. The domain model consists of a class diagram and a set of constraints expressed formally with the Object Constraint Language (OCL) [15]. The main challenge in the design of this domain model is how to capture properly the concepts and their relationships and model exactly what is an AMF configuration, not more but not less?

During this process, we have been tempted to capture into the domain model every single aspect, including runtime ones, either in the class diagram and/or using OCL constraints. The close interaction with the domain expert allowed us to avoid some pitfalls that would have led to over- or under-specification. The consequences would be a profile that excludes perfectly valid AMF configurations due to over-specification of the requirements including runtime characteristics or that accepts invalid AMF configurations because of configuration requirements not taken into account.

Our main contributions in this paper are: (1) we provide an overview of the domain of availability management as it is defined in the SAF specifications; (2) we briefly describe the domain model of our UML profile; (3) we discuss some of the challenges encountered in the process of designing the domain model; and finally (4) we report on the usage of the domain model for the validation of AMF configurations.

The remaining part of this paper is structured as follows. In Section 2, we introduce some important AMF concepts used throughout the paper. In Section 3, we discuss the main issue of categorizing AMF requirements into configuration time versus runtime, and present a compilation of sample AMF requirements used throughout the paper. In Section 4, we describe the main characteristics of the domain model and we discuss some design decisions. In Section 5, we share the main lessons learned with respect to the under/over-specification problem through specific examples. We briefly discuss the usage of the domain model for the validation of AMF configurations in

Section 6. In Section 7, we review related work before concluding the paper in Section 8.

2. AMF Specification and Concepts

SAF has developed the Application Interface Specification (AIS), which specifies a set of services including AMF. The AMF specification defines an API for availability management and an information model. This model describes the organization of the resources and services, the different *entities* to be managed by AMF in a running system; the *types* of these entities that describe common features of the entities belonging to them; and the cluster nodes on which the entities are deployed.

2.1 AMF Entities and Entity Types

The basic entity of an AMF configuration is an AMF *component*, which represents a set of software and/or hardware resources that can provide some basic services referred to as *component service instances* (CSIs). The components are logically grouped into *service units* (SU) in order to combine their functionality into higher level services referred to as *service instances* (SIs). In order to protect these services using redundancy, SUs are grouped into *service groups* (SGs). An SG protects a set of SIs that are assigned to its SUs in different roles. When a particular SI is assigned to an SU, its composing CSIs are assigned to the components in the SU. The grouping of service groups forms an AMF *application*. From a deployment perspective, each SU is deployed on an AMF *node*, thus an SG is deployed on a node group. The set of all AMF nodes forms the AMF *cluster*.

The notion of type is introduced in the AMF specification to capture common characteristics shared by all the entities that belong to the same type. In addition, the types define also the relation between entities. For example the SU type specifies the set of component types it contains, which defines components of what types must compose each of the SUs of the SU type. However, not all the entities are typed. The typed entities (and their corresponding types) are: the application (application type), the service group (SG type), the service unit (SU type), the component (component type), the service instance (service type), and the component service instance (CS type). The *non-typed* entities are: the cluster and the node. In a complete AMF configuration each typed entity should refer to its type.

2.2 Redundancy Models

AMF coordinates the redundant entities (SUs and their components) of an SG according to a certain redundancy model. This defines the number of active SUs (respectively components), the number of standby SUs (respectively components) to

protect an SI (respectively CSI). For each SI AMF selects at runtime which SU shall act in which role and makes the appropriate assignments via API callbacks to the components. In the AMF specification [3], several redundancy models have been defined. These are the 'No redundancy', 2N, N+M, N-Way, N-Way-Active redundancy models. For instance, in an SG protecting a set of SIs according to the 2N model, at most one SU can be active for all the SIs, and at most one other SU can be standby for all the SIs protected by the SG.

2.3 Component Capability Model

Within an SU, each component has its component capability model which is defined as a triple (x, y, b), where x represents the maximum number of active CSI assignments and y the maximum number of standby CSI assignments the component can have for a particular component service type, and b determines whether it can support these roles simultaneously or not. The redundancy model used by an SG should be consistent with the capabilities of the components of its SUs. For example, a component that can have only one active or only one standby CSI assignment at a time (1, 1, false) can be used in an SG with a 2N redundancy model, but it is not *valid* for a SG with an N-way redundancy model, where each of the SUs may be active for some SIs and standby for others simultaneously, e.g. (n, m, true).

2.4 AMF Configuration Example

Figure 1 shows an example of configuration of AMF entities. In this example, a cluster is composed of two nodes (A and B). It hosts an Application consisting of one SG protecting one SI in a 2N redundancy model. The SG consists of two SUs, SU1 and SU2, each composed of two components. The distribution of the active and standby assignments is shown in this figure. However, it is not part of the configuration as defined by AMF. These assignments are decided by an AMF service implementation at runtime.

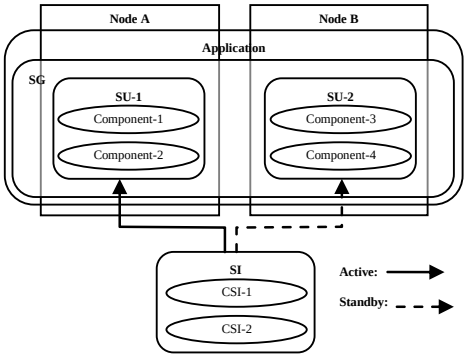

Fig. 1. Example of AMF Configuration

3. Categorizing AMF Requirements

As the AMF specification is about defining what a valid AMF configuration is and how it is manipulated at runtime by a compliant AMF service implementation, the first step in our process is to distinguish clearly between configuration time and runtime requirements. However, this is not straightforward and instead of two categories, the requirements defined in the AMF standard specification can be organized into three categories as shown in Figure 2. The first category, configuration requirements, clearly encloses the requirements defining what an AMF configuration is. These constraints can be checked at configuration time. They should be reflected in the domain model in the class diagram or using OCL constraints. The second category of requirements, run-time requirements, is clearly related to the dynamic behavior of the AMF service and hence need to be satisfied by any AMF-compliant middleware. These requirements are out of the scope of our profile and therefore should not be captured either in the class diagram or using OCL constraints.

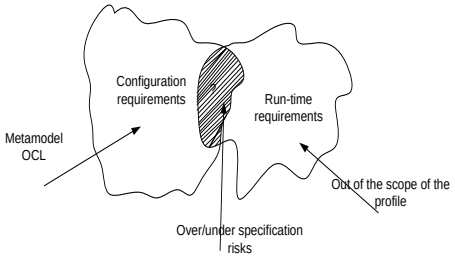

Fig. 2. AMF Configuration vs. Run-time Requirements

The challenging aspect regarding the AMF requirements stems from the third category, which is depicted by the overlapping region in Figure 2. Often the specification does not provide a clear cut as whether these are configuration requirements or AMF service runtime related requirements. As a configuration defines relations between the different entities involved, there is a temptation to define all of them at configuration time. This is wrong as some of these relations are defined only at runtime to allow more flexibility to the middleware implementing the AMF service. Some of these relations defined at runtime are however based on other related configuration time constraints to ensure that the configured application will provide and protect the service independently from the decisions taken by the AMF service implementation. Capturing and specifying these configuration time constraints without the related runtime relationships between the entities is not straightforward. Moreover, it is not clear which ones and to what extent these should be captured in the domain model. Indeed, here we are facing the traditional over- vs. under-specification problem. Over-specification occurs when we try to capture some requirements in our domain model but these requirements are not configuration time and related to the runtime behavior of the AMF service and to its manipulation of the configuration. On the other hand, under-specification occurs when we mistakenly

consider a requirement as not checkable at configuration time. The consequence of such misinterpretations is a profile that may exclude valid AMF configurations when we over specify the requirements and/or that includes invalid configurations when all configuration time requirements are not captured. In Section 5, we elaborate more on this issue with specific examples.

Table 1 provides a set of requirements, taken from the standard, that we will use in the next sections to illustrate different issues and some design decisions. In the column Configuration/Run-time, we use the letter "C" to identify the requirements that we deem belong to the configuration requirements subset and, hence, can be checked in the configuration. The two other columns are used to indicate whether the requirement is captured in the class diagram of the domain model, using OCL constraints, or both. These issues are discussed further in the next sections. We use the letter "R" for the requirements related to the dynamic behavior of the AMF service and therefore are out of the scope of our profile. For the remaining set of requirements, identified with "?", it is not clear how to define the constraints that would allow to check their satisfaction at configuration time without a risk of over-/under-specification.

4. An Overview of the Domain Model for AMF Configurations

The AMF profile is organized into packages for distinguishing between service, service provider and deployment concepts as defined in the standard. The service package is for the description of the concepts related to the definition of the services to be provided; the service provider package is for the description of the (logical and physical) resources defined in a system to provide services and their organization in terms of composition of their functionalities or in terms of redundancy. The AMF specification emphasizes the idea of service and service provider separation to enable the moving of services around service providers. The deployment package is for the description of deployment concepts such as clusters and nodes. Our profile overall architecture captures this idea as shown in Figure 3

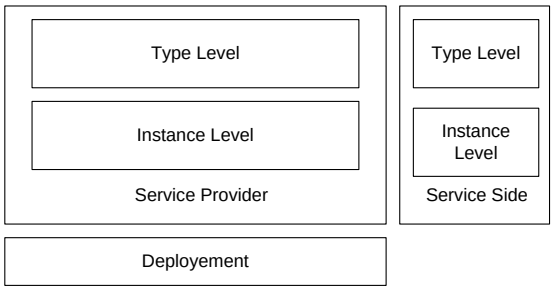

Fig. 3. AMF Profile Architecture

Table 1. A Sample of AMF Requirements

	#	Text	Reference in AMF-B.03.01	Configuration/ runtime	Class Diagram	OCL
General	RQ1	*An SU can contain any number of components, but a particular component can be configured in only one SU.*	Sec: 3.2.3.1 Page: 41	C	✓	
	RQ2	*Local components and external components cannot be mixed within a service unit. Local service units can contain only local components and External service units can contain only external components.*	Section: 3.2.4 Page: 41	C	✓	
	RQ3	*The service unit type defines a list of component types and, for each component type, the number of components that a service unit of this type may accommodate.*	Section: 3.2.4.1 Page: 42	C	✓	
	RQ4	*A CSI can be assigned to a given component only if the component configuration indicates that the component supports this particular type of CSI*	Section: 3.2.5 Page: 43	?		
	RQ5	*If node groups are configured for both the SUs of an SG and the SG, the nodes contained in the node group for the SU can only be a subset of the nodes contained in the node group for the SG.*	Section: 3.7.1.2 Page: 92	C		✓
Container/Contained Components	RQ6	*A container component and all its associated contained components must reside on the same AMF node.*	Section: 3.2.2.1.1 Page: 35	?		
	RQ7	*The termination of a container component implies the termination of all associated contained components. The termination of a contained component does not imply the termination of either the associated container component or the collocated contained components.*	Section: 3.2.2.1.1 Page: 35	R		
	RQ8	*A contained component must not be a proxy component.*	Section:3.2.2.3 Page: 37	C	✓	
	RQ9	*A contained component must be configured with the container CSI.*	Section: 3.2.3 Page: 40	C	✓	
	RQ10	*All contained components in an SU must have the same associated container component, and this association is achieved by the usage of a single container CSI*	Section: 3.2.4 Page: 42	C		✓
	RQ11	*An SU that contains a contained component can only contain collocated contained components*	Section: 6.1.5 Page: 189	?		
	RQ12	*Container components and contained components must not be located in the same SU.*	Section: 6.1.5 Page: 189	?		
Pre-instantiability	RQ13	*All SA-aware components are pre-instantiable components.*	Section: 3.2.2.4 Page: 39	C		✓
	RQ14	*All non-proxied, non-SA-aware components are non-pre-instantiable components.*	Section: 3.2.2.4 Page: 39	C		✓
	RQ15	*Proxied Component might be pre-instantiable or not*	Table:3 Page: 39	C		
Proxcy/ Proxied	RQ16	*A proxied component must be configured with the proxy CSI that provides "proxying".*	Section: 3.2.3 Page: 40	C	✓	✓
	RQ17	*A proxy component and its pre-instantiable proxied component must not reside in the same SU*	Section: 3.2.4 Page: 42	?		
Redundancy Capability / Model	RQ18	*To participate in an SG, all components in the SU must support the capabilities required for the redundancy model defined for the SG.*	Section: 3.2.6 Page: 44	C		
	RQ19	*The only redundancy model supported for service groups having service units containing a container component is the N-Way active redundancy model*	Section: 6.1.6 Page: 190	C		✓

The domain model of the AMF profile is defined using a class diagram complemented with a set of OCL constraints to capture the configuration requirements. The requirement RQ 2 in Table 1, for instance, is captured explicitly in

the model using a composition relationship between local service unit class and local component class. This is highlighted in the dashed region A in Figure 4. Other requirements, such as RQ 19 in Table 1, are more complicated to be captured in the class diagram. Indeed, RQ 19 involves a relationship between components and SG, which is not explicit in the class diagram. The requirement is therefore expressed in OCL as follows:

- **context** MagicAmfContainerComponent
  inv: self.magicAmfLocalComponentMemberOf.
  magicSaAmfSUMemberOf.
  oclIsKindOf(MagicAmfN-WayActiveSG)

The domain model reflects component categories as defined in the specification, namely SA-aware, container, contained, proxy, proxied, local, external and pre-instantiable components. The class diagram in Figure 4 shows the hierarchy of components for the first seven categories. We model pre-instantiable components differently; we use a specific configuration attribute, *magicAmfCtIsPreinstantiable*, at component type level (i.e. in the MagicSaAmfComponentType class as shown in Figure 4) instead of using a specific class for this category of components. The rationale behind our decision is that modeling pre-instantiable component category with a specific class would have required a further specialization of the local proxied components into a new class that represents non-pre-instantiable components. This is indeed necessary to account for the requirement RQ 15 in Table 1. Obviously, this would have complicated further the component hierarchy. As a consequence of our modeling of the pre-instantiable component category, the requirements RQ 13 and RQ14 in Table 1 are captured in OCL using this new configuration attribute as follows:

- **context** MagicSaAwareCompType
  inv: self.magicSaAmfCtIsPreinstantiable = true

- **context** MagicAmfNon-ProxiedNon-SaAwareCompType
  inv: self.magicAmfCtIsPreinstantiable = false

5 Over/Under-specification of the Domain Model

In this section, we discuss some aspects of AMF configuration constraints over-/under-specification, which we have encountered during the process of defining the domain model. We limit, however, this discussion to only two situations because of the lack of space.

5.1 Component Capability Model

The capability of a component, defined in the standard and introduced in Section 2, depends on the target component service type (CsType). Therefore, we capture the component capability model using two association classes as shown in the Figure 5.

The first association class, called *MagicSaAmfCtCsType,* captures the fact that a component type might support several CsTypes, each with different capability. The second is to capture that the capability of a particular component of a component type is further restricted with respect to a certain CsType by putting limitations on the number of active and standby that a component can take. In our domain model, this second association class is called *MagicSaAmfCompCsType*.

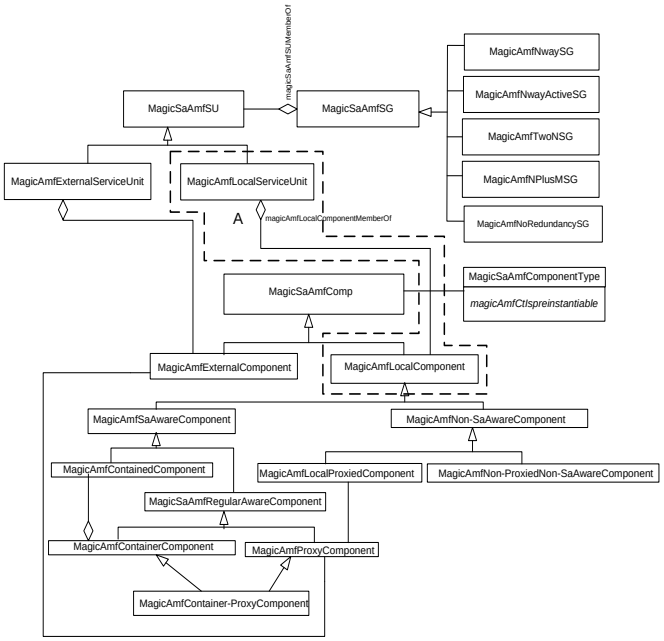

Fig. 4. Partial Domain Model

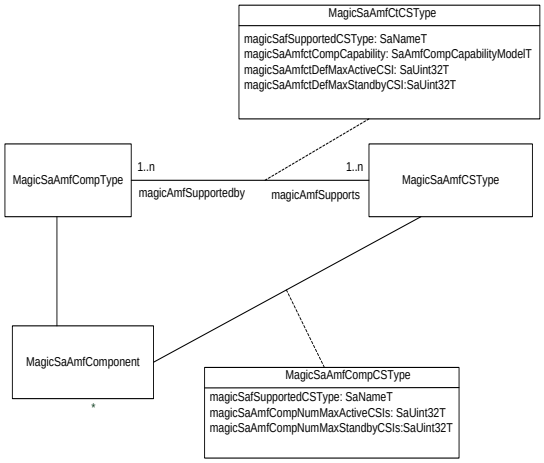

Fig. 5. Component Capability Model

The issue related to the component capability shows up when the components grouped into an SU support overlapping sets of CsTypes and, in addition, the capability models of the components with respect to the common supported CsTypes are different. As stated in RQ 4 in Table 1, an AMF service implementation uses the configuration time relationship between components and CsTypes to assign CSIs to components. However, this assignment of CSIs to components happens only at runtime and under the control of an AMF service implementation. Different behaviors/runtime decisions of AMF for the same configuration, i.e. different assignments of CSIs to components, may lead to different availability levels. The configuration is not the right place where to control the assignments and therefore the level of availability obtained from AMF and this is not a configuration time decision

We were tempted to capture and fix in our domain model these assignments at configuration time. This was wrong, and this over-specification has been avoided with the help of the domain expert.

5.2 Proxy and Proxied Components

Several requirements in the AMF specification relate the proxy and their proxied components. For instance, the requirement RQ 17 in Table 1 specifies a location constraint between a proxy and a proxied component. In the initial version of our domain model, we related formally proxy and proxied components with an association as shown by the dashed association (A) in Figure 6. The interactions with the domain expert showed that this relationship is not a configuration time relationship and it is only at run-time that an AMF service implementation selects and assigns a particular proxy component to a particular proxied component according to the requirement RQ 4 in Table 1. This association is therefore removed from our model as it represents a typical case of over-specification, which fixes runtime relationships at configuration time.

The requirement RQ 16 on the other hand specifies a configuration time relationship between a proxy and a proxied component through the proxyCSI. This, however, needs to be captured in our model and this is achieved with the association end *magicSaAmfCompProxyCSI* of the association between MagicLocalProxiedComponent and MagicSaCSI classes as shown in Figure 6. This is a particular CSI through which a proxy component is assigned the task of "proxying" a particular proxied component. Consequently, the constraints on the proxy-proxied relationship, such as the requirement RQ 17, 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 a proxy component that is able to be the proxy component for the proxied component. This constraint translates to the existence of a proxy component that support a component service type (CsType) to which the proxyCSI of the proxied component in question belongs. The domain model should capture this constraint otherwise we fall into an under-

specification case and may mistakenly consider as valid, configurations that do not satisfy AMF requirements. Several other situations like the ones related to container and contained component as shown in Figure 6 (A) and (B) are not discussed here due to space limitations.

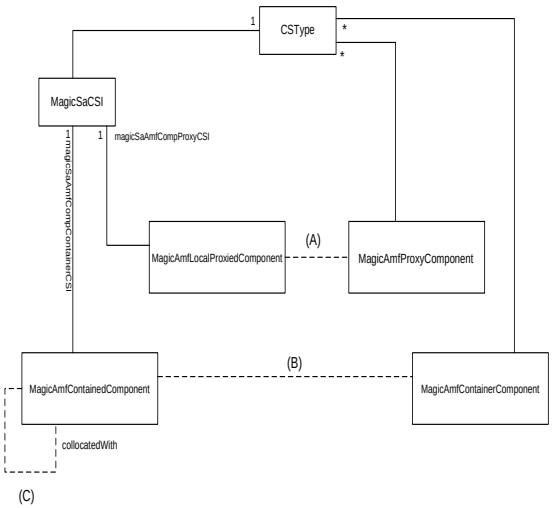

Fig. 6. Proxy-proxied Component Relationship

6. An Application: AMF Configuration Validation

A straightforward application of the domain model is the validation of AMF configurations. These are often built manually and deployed in a SAF system through the Information Model Management (IMM) service [13] using the IMM XML format [14]. The design of an AMF configuration is a tedious and error prone task due to the large number of AMF requirements that have to be taken into consideration. Consequently, checking the compliance of AMF configurations against the AMF specification is crucial.

We have implemented a prototype tool for the validation of AMF configurations. The designer creates an AMF configuration in the IMM XML standard format. An AMF model instance is created from the IMM XML file. The validation starts by mapping this instance into an instance of our domain model, and then in a second step the OCL constraints are checked against this later instance as illustrated in Figure 7.

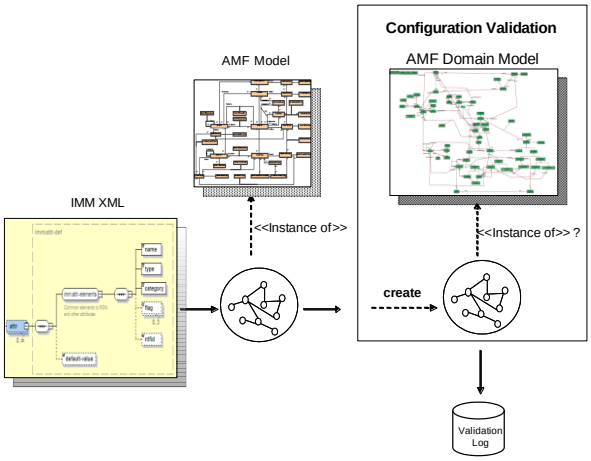

Fig. 7. Configuration Validation Tool

7. Related Work

One of the UML profile standardized by the OMG [7] and which is related to our research work is described in [8]. The study of this profile revealed however that it cannot be mapped to the concepts introduced in the AMF standard specification. Consequently, we could not use it as a leverage to define our profile. Another profile intended to support the modeling and analysis of reliability and availability is described in [9]. This profile is completely unrelated to the concepts defined in the AMF specification and hence cannot be used to support the modeling and analysis of AMF configurations. As mentioned earlier in this paper, the AMF specification defines what an AMF configuration is and the behavior expected from an AMF service implementation. This has not been the common path for most of the existing profiles that focus on formalizing general availability and dependability concepts.

The work reported in [10] is related to this paper. The authors describe an approach based on MDA to generate AIS configurations. They present in particular a platform independent model (PIM) for AIS configurations. Such PIM is not, however, supported with a specific UML profile that would provide a comprehensive coverage of AMF concepts.

An UML profile related to HIDENETS [11] architecture and services is presented in [12]. The metamodel of this profile relates HIDENETS artifacts to some SAF AIS services using “façade objects”. This metamodel does not specify any of the concepts relevant to the AMF service or AMF configurations.

8. Conclusions

In this paper, we discussed our work in designing a profile for AMF configurations. We briefly introduced our approach and current results before elaborating on the challenges of capturing only necessary requirements into the domain model. These challenges are mainly due to the fact that the AMF standard specification defines simultaneously what a valid AMF configuration is and what the expected behavior from an AMF service implementation is. An AMF configuration is defined at configuration time, while an AMF service implementation decides at run-time of several assignments and relationships.

In our initial attempts in defining the domain model we have been tempted to capture more than what is required and go beyond configuration time requirements. Several iterations with the domain expert have been necessary to distinguish between the different categories of requirements in the AMF specification. Dropping the runtime requirements from the domain model has led to other difficulties in specifying related configuration requirements that are necessary for the definition of AMF configurations.

Our goal is to enable the rigorous modeling of AMF configurations and their analysis. The analysis package will be investigated and the dynamic behavior of the AMF service will be part of this package in order to enable the analysis of AMF configurations.

References

1. A. Avizienis, J.C. Laprie, B. Randell and C. E. Landwehr: Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Security Computing 1(1): pp. 11--33 (2004)
2. Gary Audin: Reality Check On Five-Nines. Business Communications Review, pp 22--27, (2002)
3. The Service Availability Forum, http://www.saf.com
4. Service Availability Forum: Application Interface Specification. Overview SAI-Overview-B.04.01.
5. Service Availability Forum, Application Interface Specification. Availability Management Framework SAI-AIS-AMF-B.03.01.
6. Bran Selic: A Systematic Approach to Domain-Specific Language Design Using UML. Proceedings of the 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC 2007), pages 2--9. IEEE Computer Society, (2007)
7. Object Management Group (OMG), http://www.omg.org/
8. OMG: UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms, Version 1.1, OMG Document Number: formal/2008-04-05 (2008)
9. Simona Bernardi, José Merseguer: A UML profile for dependability analysis of real-time embedded systems. Proceedings of the 6th International Workshop on Software and Performance, WOSP (2007)

10. A. Kövi, D. Varró: An Eclipse-Based Framework for AIS Service Configurations. ISAS'2007. LNCS, vol. 4526, pp. 110–126. Springer (2007)
11. HIDENETS Research Project, http://www.hidenets.aau.dk/
12. HIDENETS: A UML Profile and Design Pattern Library, Deliverable D 5.1 (2007)
13. Service Availability Forum, Application Interface Specification. Information Model Management Service SAI-AIS-IMM-A.02.01
14. SAI-AIS-IMM-XSD-A.01.01.xsd
15. OMG: Object Constraint Language, Version 2.0, OMG Available Specification: formal/06-05-01 (2006)
16. OMG, Unified Modeling Language (OMG UML), Infrastructure, V2.1.2, OMG Document Number: formal/2007-11-04 (2007)