

A Model Based Framework for Service Availability Management

by

Pejman Salehi

A Thesis

in

The Department of Electrical and Computer Engineering

Presented in Partial Fulfilment of the Requirements

for the Degree of Doctor of Philosophy at

Concordia University

Montreal, Quebec, Canada

April 2012

© Pejman Salehi, 2012

ABSTRACT

A Model Based Framework for Service Availability Management

Pejman Salehi, Ph.D.

Concordia University, 2012

High availability of services is an important requirement in several domains, including mission critical systems. The Service Availability Forum (SA Forum) is a consortium of telecommunications and computing companies that defines standard middleware solutions for high availability. Availability Management Framework (AMF) manages the high availability of services by coordinating their application components according to redundancy models. To protect these services, AMF requires a configuration, i.e. a representation of the organization of the logical entities composing an application under its control. AMF configuration design is error-prone and tedious if done manually, due to the complexity of the AMF domain. This PhD thesis explores the effective design and analysis of AMF configurations, proposing a model-based management framework that facilitates this process. We propose a domain-specific modeling language that captures AMF domain concepts, relationships, and constraints, facilitating the management of AMF configurations. We define this language by extending UML through its profiling mechanism, capturing the concepts of AMF configurations and the description of the software for which the configuration will be generated.

We introduce a new approach for the automatic generation of AMF configurations based on our UML profile using model transformation techniques. This approach consists of a set of transformations from the software description entities into AMF configurations

while satisfying the requirements of the services to be provided as well as the constraints of the deployment infrastructure.

We also propose a third-party AMF configuration validation approach consisting of syntactical and semantic validations. Syntactical validation checks the well-formedness of third-party configurations by validating them against AMF standard specification requirements captured in our UML profile. Semantic validation focuses on ensuring the runtime protection of services at configuration time (the SI-Protection problem). SI-Protection has combinatorial aspects and results in an NP-hard problem for most redundancy models, which we have tackled by devising a heuristic-based method, overcoming its complexity.

We present proofs of concepts by using different available technologies: IBM Rational Software Architect (RSA) for implementing our UML profiles, Eclipse environment for developing a prototype tool for validating third-party configurations, and Atlas Transformation Language (ATL) for developing a prototype implementation of our model-based configuration generation approach.

Acknowledgments

I would like to thank the many people who have made this thesis possible through their wisdom and mentorship: First, my supervisors Dr. Ferhat Khendek and Dr. Abdelwahab Hamou-Lhadj who have guided me through this venture with their insightful advice; and secondly, to Dr. Maria Toeroe for her tireless dedication and meticulousness. I would also like to express my gratitude to my colleagues and friends from the MAGIC Project, Dr. Pietro Colombo, Ali Kanso, and Dr. Abdelouahed Gherbi, for their friendship and support.

This dissertation would not have been possible without the generous financial support received through the organizations contributing to the MAGIC Project: the Natural Sciences and Engineering Research Council (NSERC) of Canada, Ericsson Software Research and Concordia University. I would also like to express my appreciation for the research facilities provided by Concordia University.

I would like to extend my thanks to the examining committee for their support during the various stages of my PhD and for their effort in the final evaluative process.

I express my profound appreciation to my girlfriend, Katherine, for her love, encouragement, support and her tireless proofreading.

I would like to show my gratitude to my little sister Mahsa for always supporting me during the various stages of my life. I would also like to thank my uncles for always believing in me and encouraging me during my studies.

I owe my deepest gratitude to my parents, Mahnaz and Hossein, to whom this thesis is dedicated, for their endless love and support throughout my life. I would like them to know that I am eternally indebted to them.

Table of Contents

1	Introduction	1
1.1	Thesis Motivation.....	1
1.2	Contributions.....	4
1.3	Thesis Organization.....	6
2	Background and Literature Review	8
2.1	High Availability and SA Forum	8
2.1.1	Service Availability	8
2.1.2	The Service Availability Forum.....	9
2.1.3	The Availability Management Framework.....	10
2.1.4	The Entity Types File.....	17
2.2	Modeling and UML Profiles	18
2.2.1	The UML Profiling Mechanism.....	19
2.2.2	Related UML Profiles.....	22
3	Modeling Framework- Domain Models	28
3.1	Domain Modeling Process	29
3.2	AMF Domain Model	30
3.2.1	AMF Components and Component Types	31
3.2.2	SU, SG, SI, CSI and their Types.....	33
3.2.3	Deployment Entities.....	34
3.2.4	Well-formedness Rules.....	35
3.2.5	Challenges.....	40
3.3	ETF Domain Model	41

3.3.1	Basic Service Provider and Service Elements	42
3.3.2	Compound Elements	44
3.3.3	Software Dependency	46
3.3.4	Domain Constraints	47
3.3.5	Challenges	48
3.4	CR Domain Model	49
3.5	Summary	51
4	Modeling Framework- Mapping to UML Metamodel	52
4.1	Mapping Domain Model Concepts to UML Metaclasses	54
4.1.1	AMF Component	55
4.1.2	AMF Service Unit (SU)	55
4.1.3	AMF Service Group (SG)	55
4.1.4	AMF Application	56
4.1.5	AMF Component Service Instance (CSI)	56
4.1.6	AMF Service Instance (SI)	57
4.1.7	AMF Node	57
4.1.8	AMF Cluster and AMF NodeGroup	58
4.1.9	AMF Entity Type Elements	58
4.1.10	ETF Types	63
4.1.11	CR Elements	65
4.2	Mapping the Domain Relationships to the UML Metamodel	66
4.3	Specifying Constraints	71
4.3.1	Constraints on Relationships	72
4.3.2	Constraints on Metaclasses	73
4.4	Challenges	73
4.5	Summary	74

5	AMF Configuration Validation	76
5.1	Syntactical Validation of AMF Configurations	76
5.2	Semantic Validation of AMF Configurations	77
5.2.1	Definitions and Notations	78
5.2.2	Service Instance Protection for the 2N and No-Redundancy Models	81
5.2.3	Service Instance Protection for the N+M Redundancy Model	84
5.2.4	The N-Way-Active and N-Way Redundancy Models	89
5.2.5	Overcoming Complexity for Special Cases	91
5.2.6	Overcoming Complexity with Heuristics: Checking for Service Protection Using Heuristics	95
5.3	Summary	108
6	Model-based AMF Configuration Generation	110
6.1	Overall View	110
6.2	ETF Type Selection.....	117
6.2.1	CSITemp Refinement	119
6.2.2	SITemp Refinement	125
6.2.3	SGTemp Refinement	127
6.2.4	Dependency Driven Refinement.....	129
6.2.5	Completing the Refinement	132
6.3	AMF Entity Type Creation	137
6.3.1	AMF SGType and AppType Generation.....	140
6.3.2	AMF SUType and SvcType Generation.....	144
6.3.3	AMF Component Type and CSType Generation	150
6.4	AMF Entity Creation.....	154
6.4.1	Step 1: AMF Entity Instantiation.....	156
6.4.2	Step 2: Generating Deployment Entities.....	164

6.4.3	Step 3: Finalizing the Generated AMF Configuration.....	165
6.5	Limitations	165
6.6	Summary	166
7	Implementation of the Framework and Application.....	168
7.1	Implementation of the Model-based Framework.....	168
7.2	The Online Banking System	170
7.2.1	The Billing Service	171
7.2.2	The Authentication Service.....	172
7.2.3	The Money Transfer Service	173
7.2.4	Web Server and User Interface.....	174
7.2.5	Database Management System	176
7.2.6	General Inquiries.....	177
7.2.7	Transaction Information.....	178
7.2.8	SUType Level Dependency	179
7.3	Configuration Requirements for the Online Banking System	180
7.4	Generation of an AMF Configuration for Safe Bank Online Banking System	188
7.4.1	Selecting ETF Types.....	188
7.4.2	Creating AMF Types	191
7.4.3	Creating AMF Entities	194
7.5	Validation of the Model-based AMF Configuration Generation Approach	197
8	Conclusion and Future Work.....	202
8.1	Conclusion.....	202
8.2	Future Research.....	204
8.2.1	Model-based AMF Configuration Generation.....	204
8.2.2	Performance Evaluation of Heuristics Based Validation Approach.....	205

8.2.3	Bridging the Gap between User Requirements and Configuration Requirements	206
8.2.4	UML Profiling	206
8.2.5	Model-driven Software Development.....	207
	Bibliography	210
	Appendix I	214

List of Figures

Figure 1-1 Overview of the AMF configuration management framework.....	5
Figure 2-1 The Service Availability Interfaces.....	10
Figure 2-2 Redundancy models defined in the AMF specification	13
Figure 2-3 An example of an AMF configuration	16
Figure 2-4 An example of ETF model.....	18
Figure 3-1 Domain Modeling Process	30
Figure 3-2 AMF Component Categories	31
Figure 3-3 AMF Component Type Categories	32
Figure 3-4 Service Unit and Service Group Categories.....	33
Figure 3-5 Component Service Instance and Service Instance.....	34
Figure 3-6 AMF Nodes, Node Groups, and Cluster	34
Figure 3-7 Relationship of CStype with component and component type.....	36
Figure 3-8 Component Type and CStype Categories.....	43
Figure 3-9 Compound elements.....	45
Figure 3-10 Configuration Requirement (CR) domain model.....	49
Figure 4-1 The process of mapping to the UML metamodel and concrete syntax definition.....	53
Figure 4-2 Relationship between AMF SI and AMF CSI	69
Figure 5-1 Architecture of Validation Tool	77
Figure 5-2 Complexity of the SI-Protection for the N+M redundancy model	87
Figure 5-3 Incremental AMF configuration design using BF method with relative capacity sorting criterion.....	104

Figure 5-4 An example for the incremental design approach.....	107
Figure 5-5 Overview of the incremental design approach.....	108
Figure 6-1 The overall process of model-based AMF configuration generation.....	111
Figure 6-2 The main phases of the model transformation approach	111
Figure 6-3 The relation between the models and the transformation phases.....	113
Figure 6-4 The transformation steps for ETF Type Selection phase	117
Figure 6-5 The result of the ETF Type Selection from the metamodel perspective.....	118
Figure 6-6 The activity diagram describing the selection of ETF Component Types....	120
Figure 6-7 The activity diagram describing the selection of ETF SUTypes	125
Figure 6-8 The activity diagram describing the process of selecting ETF SGTypes	128
Figure 6-9 The transformations performed to complete the refinement phase.....	133
Figure 6-10 The result of the AMF Entity Type creation phase from the metamodel perspective	138
Figure 6-11 The transformation steps of the AMF entity type creation phase	139
Figure 6-12 The result of the AMF Entity creation from the metamodel perspective ...	155
Figure 6-13 The flow of transformations to generate AMF entities.....	157
Figure 7-1 ATL Transformation scheme	170
Figure 7-2 ETF model for billing part of an online banking software bundle.....	172
Figure 7-3 ETF model for the authentication part of an online banking software bundle	173
Figure 7-4 ETF model for money transfer part of online banking software bundle.....	174
Figure 7-5 ETF model for web server part of online banking software bundle	175
Figure 7-6 ETF model for user interface part of online banking software bundle	176

Figure 7-7 ETF model DBMS part of online banking software bundle	177
Figure 7-8 ETF model for the general inquiries part of an online banking software bundle	178
Figure 7-9 ETF model for the transaction information part of an online banking software bundle.....	179
Figure 7-10 SUType level dependency.....	180
Figure 7-11 The SGTemplates of the Safe Bank online banking system.....	181
Figure 7-12 Configuration requirement elements of WebModules and WebServer SGTemplates.....	184
Figure 7-13 Configuration requirement elements of Security, Information, and DB SGTemplates.....	184
Figure 7-14 Configuration requirement elements of Banking SGTemplate.....	185
Figure 7-15 Configuration requirements for deployment infrastructure	187
Figure 7-16 ETF Type selection phase for the DBMS part of online banking ETF.....	189
Figure 7-17 ETF Type selection phase for TransactionManagement SITemplate.....	190
Figure 7-18 AMF Type creation phase for the DBMS part of online banking configuration.....	191
Figure 7-19 AMF SGType, AMF SUType, and AMF SvcType generation steps for TransactionManagement SITemplate and Banking SGTemplate.....	192
Figure 7-20 AMF Component Type and AMF CSType generation steps for the CSITemplates of TransactionManagement SITemplate.....	193
Figure 7-21 Created AMF Types for the transaction management part of online banking configuration.....	194

Figure 7-22 AMF entity creation phase for the DBMS part of online banking configuration..... 196

List of Tables

Table 4-1 The summary of the stereotypes defined for AMF entities and entity types....	59
Table 4-2 The summary of the stereotypes defined for ETF types.....	64
Table 4-3 The summary of the stereotypes defined for CR elements.....	66
Table 4-4 Summary of Stereotypes Related to the Relationships between Domain Concepts.....	70
Table 6-1 The list of the associations that model the relationships among elements of the sub-profiles	114
Table 6-2 The list of additional attributes.....	117
Table 7-1 List of values of attributes of the SGTemplates specified for the Safe Bank online banking system.....	182
Table 7-2 List of the values of attributes of SITemplates and CSITemplates of WebModules and WebServer SGTemplates	183
Table 7-3 List of the values of attributes of SITemplates and CSITemplates of Security, Information, and DB SGTemplates	185
Table 7-4 List of the values of attributes of SITemplates and CSITemplates of Banking SGTemplates.....	186

List of Equations

Equation 2-1 System availability	9
Equation 5-1 Adding capacity lists	79
Equation 5-2 Comparison of capacities	79
Equation 5-3 Division between capacities	79
Equation 5-4 Active and Standby relation between a set of SUs and a set of SIs	80
Equation 5-5 Operators for active/standby relation	81
Equation 5-6 Formal specification of the 2N redundancy model	83
Equation 5-7 Necessary and sufficient conditions for the 2N redundancy model	83
Equation 5-8 Formal specification of the No-redundancy model	84
Equation 5-9 Formal specification of the N+M redundancy model	85
Equation 5-10 Formal specification of the N-Way-Active redundancy model	90
Equation 5-11 Formal specification of the N-Way redundancy model	90
Equation 5-12 Active/Standby capacity of an SU w.r.t. to an SI	92
Equation 5-13 Necessary and sufficient conditions for the N+M redundancy model	93

List of Acronyms

AIS	Application Interface Specification
AMF	Availability Management Framework
ATL	Atlas Transformation Language
CSI	Component Service Instance
CSType	Component Service Type
DSL	Domain Specific Language
DSML	Domain Specific Modeling Language
EMF	Eclipse Modeling Framework
ETF	Entity Types File
HA	High Availability
IMM	Information Model Management
MDA	Model Driven Architecture
OCL	Object Constraint Language
OMG	Object Management Group
RSA	Rational Software Architect
SA Forum	Service Availability Forum
SG	Service Group
SGType	Service Group Type
SI	Service Instance
SU	Service Unit
SUType	Service Unit Type

SvcType	Service Type
UML	Unified Modeling Language
XMI	XML Metadata Interchange

Chapter 1

Introduction

1.1 Thesis Motivation

The growing reliance on computing platforms has led to an increase in the customer's demand for robust and safe systems. For such systems, the requirement of providing services with minimal to no interruptions has become essential. The development of highly available (HA) systems has been investigated for several years and different solutions have been proposed (e.g. [Lomb 1996, Vogels 1998, Watts 2007]). However, these solutions are proprietary which hinders portability of applications from one platform to another. To address this issue, many telecommunications and computing companies have joined forces to create the Service Availability Forum (SA Forum) [SAF 2010a], a consortium that has the objective of defining standard specifications to support the development of HA systems. These standards aim to enable the portability and reusability of applications across different platforms by shifting the availability management from applications to a dedicated middleware.

One of the key SA Forum specifications is the Application Interface Specification (AIS) [SAF 2010b], which supports the development of HA applications by abstracting from their components. To achieve this, AIS defines several services, among which the most important is the Availability Management Framework (AMF) [SAF 2010d]. AMF is the

middleware service that manages the high availability of the services offered by applications by coordinating their redundant components. In order to protect 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, their types, and the deployment information.

The design of AMF configurations consists of specifying a set of elements based on the description of software entities in order to provide and protect the services as requested by the configuration designer. The description of the software entities is specified by means of Entity Types File (ETF) standard XML schema [SAF 2010e]. More specifically, the design is based on 1) the descriptions of software resources to be used as well as the description of the infrastructure supporting the deployment, 2) requirements that specify the services to be provided, and 3) other non-functional requirements such as the level of availability. The design and analysis of AMF configurations requires a good understanding of AMF entities and their relations. This is a complex task due to the following:

- The large number of entities and the numerous attributes/parameters that need to be taken into consideration.
- The large number of constraints in the standard specification. Moreover, these constraints crosscut various entities, making the process of validation extremely complex.
- Runtime versus configuration time aspects: there exist certain aspects which cannot be set at configuration time, giving the middleware the flexibility to make decisions. For instance, the AMF middleware decides to assign services to

specific service providers at runtime. However, in order to design valid configurations, the designer should predict AMF's behaviour and consider all possible assignment scenarios.

Moreover, the specifications describe the AMF configuration characteristics through the expectations of the AMF middleware at runtime. The pure configuration aspects are, therefore, rather ambiguous in the specifications, and consequently, reasoning about AMF configurations is not a straightforward process.

- The complexity of the concepts and their relationships defined in the AMF specification. For instance, the notion of types and entities is introduced to capture the limitations and capabilities on two different levels of abstractions. This increases the complexity of insuring the necessary consistency between these two levels.

Considering these complexities, a manual or an ad hoc approach for generating AMF configurations is extensively tedious and error prone. Therefore, the need for a systematic and automatic approach is inevitable. In [Kanso 2008 and Kanso 2009], Kanso et al. proposed algorithmic solutions, implemented in Java, for the automatic generation of valid AMF configurations and thus overcame the difficulties of the manual generation process. However, in using a pure code-centric method, one still needs to deal with unnecessary details and complexity at the low level of abstraction. As such, the process still remains complex and, in addition, any small changes will result in large modifications to the code.

1.2 Contributions

In this research, we address the aforementioned issues by defining a modeling framework and approaches for the design and validation of AMF configurations. The model-driven paradigm focuses on creating models, or abstractions, which are closer to particular domain concepts rather than to computing concepts [Aagedal 2005]. In this paradigm, models replace code as the primary artefacts in the development process by enabling the developers to focus on modeling the problem domain rather than on programming. Therefore, it enables the abstraction from specific programming platforms by modeling at a platform independent level. This paradigm appeared to be an appropriate solution for the specification of AMF configuration management framework. It allows methods to shift from the low levels of details to higher levels of abstraction.

The main objective of this work is to define a precise modeling framework for AMF and related approaches for design and validation of AMF configurations. More specifically, the contributions of this PhD thesis are:

- A domain specific modeling language (DSML) tailored to AMF domain concepts, semantics, and syntax. This modeling framework is designed to support the design, specification, analysis and validation of AMF configurations. We build the modeling framework by extending the Unified Modeling Language (UML). More precisely, the required DSML is represented in the form of a UML profile which integrates the concepts involved in managing an AMF configuration from creation to analysis.
- A model driven approach for the generation of AMF configurations through model transformations. This is contrasted with existing code-centric configuration

generation techniques such as the ones presented by Kanso et al. in [Kanso 2008 and Kanso 2009], and which tend to be rigid and platform-dependant.

- An approach for the validation of third-party AMF configurations. These configurations are generally built manually due to a lack of tool support for AMF. The validation process is particularly designed to address two questions: (1) Is a third-party configuration syntactically correct and well-formed with respect to the AMF standard specification? (2) Does a given AMF configuration provide the level of protection that it claims?

The modeling framework is composed of two UML sub-profiles, namely the AMF and ETF sub-profiles. The implementation of the modeling framework using proper CASE tools such as Rational Software Architect (RSA) [IBM 2011] provides us with the interface for designing and validating instances of these profiles. In other words, it provides the facilities for the AMF configuration designers or software vendors for the specification and validation of AMF configurations or ETF models.

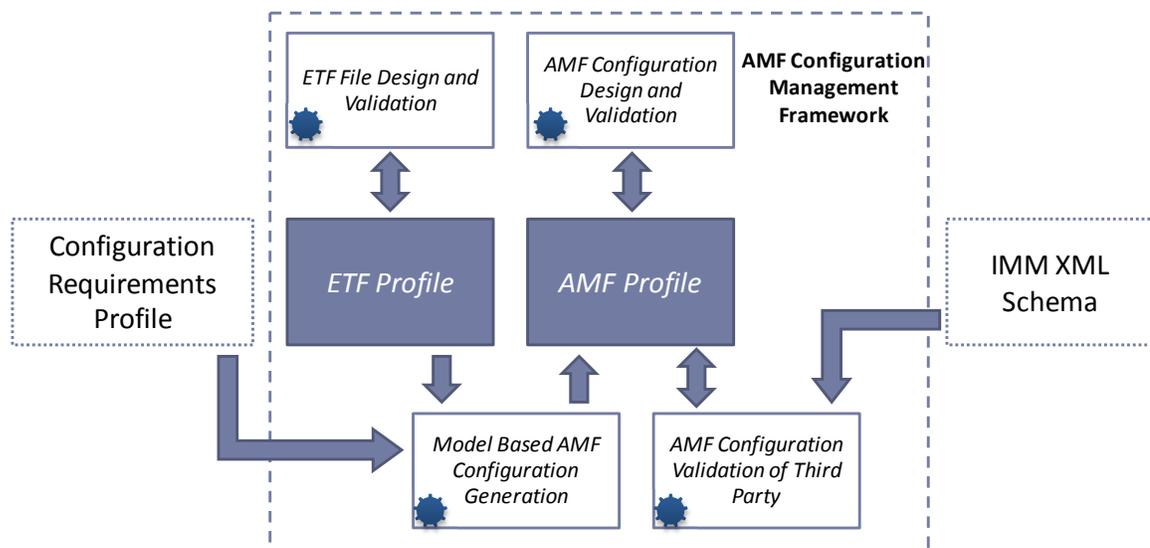


Figure 1-1 Overview of the AMF configuration management framework

Figure 1-1 illustrates the high level view of our proposed framework in which the gray squares represent elements of the modeling framework, the white squares represent the approaches, and the dashed empty squares represent external models, e.g. standard model, used by the framework. The discussion of these external models is beyond the scope of this research.

It is worth noting that the work describe in this thesis is part of a larger research project called MAGIC¹ —a collaboration between Concordia University and Ericsson Software Research— and the results of this thesis are being used in other MAGIC research streams. The term MAGIC is used throughout the profile.

1.3 Thesis Organization

The remaining parts of this thesis are organized as follows: In Chapter 2, we introduce the main concepts of high availability followed by the fundamentals of the model-driven paradigm, UML profiles, and the review of related work. In Chapter 3, we describe the domain model of our framework followed by its mappings to the UML metamodel and the description of the concrete syntax of our profile in Chapter 4. In Chapter 5, we introduce our approach for AMF configuration validation. In Chapter 6, we present and discuss our model-based approach for AMF configuration generation. In Chapter 7, we discuss the implementation of our model-driven framework. This chapter also illustrates the application of the framework through a case study for the generation of an AMF configuration for an online banking system as well as the description of all modeling

¹ MAGIC (Modeling and Automatic Generation of Information and upgrade Campaigns for service availability). <http://encs.concordia.ca/~magic/>

artefacts. In Chapter 8 we review the main contributions of this thesis and outline potential future work.

Chapter 2

Background and Literature Review

In this chapter, we explain the context of our research. More specifically, we introduce service availability, the SA Forum [SAF 2010a], and SA Forum middleware specifications focusing on AMF [SAF 2010d], and the Entity Types File (ETF) [SAF 2010e]. Model-driven paradigm is used as a general framework for the design and specification of the framework for software availability management. Therefore, in the second part this chapter, we present an overview of the main concepts of model-driven development approach. More particularly, we discuss Domain Specific Modeling Languages (DSML), Unified Modeling Language (UML), and UML's profiling mechanism. Finally, we discuss related research work focusing on existing UML profiles that capture non-functional properties of software, as well as existing approaches for the design of AMF configurations.

2.1 High Availability and SA Forum

2.1.1 Service Availability

Availability is the probability of service provision upon request, assuming that the time required for satisfying each service request is short and negligible [Wang 2005]. The availability of a system is measured in terms of the reliability of the system components

and the required time to repair the system in case of failure. It is measured using the following formula:

Equation 2-1 System availability

$$Availability = \frac{MTBF}{MTBF + MTTR}$$

in which the MTBF represents the mean time between failure (the failure rate of the system) and MTTR stands for the mean time to repair (the time to restore service) [Wang 2005]. If the availability of a system goes beyond 99.999% of the time (known as five nines), the system is considered as a highly available system.

2.1.2 The Service Availability Forum

The Service Availability Forum (SA Forum) is a consortium of several computing and telecommunications companies that develops, publishes, promotes, and provides education on open specifications in order to standardize high availability platforms [SAF 2010b]. The solution offered by the SA Forum facilitates high availability alongside service continuity.

SA Forum members have developed a set of specifications that describe various services that, when implemented, form a complete middleware for high availability. A set of APIs has also been defined in order to standardize the interface between the applications and the middleware that implements SA Forum specifications (referred to in this thesis as a SA Forum middleware). The SA Forum specifications are divided into two main groups (see Figure 2-1):

- The Application Interface Specifications (AIS) [SAF 2010b], which defines the services that handle the high availability of the application's components.
- The Hardware Platform Interface (HPI) [SAF 2010c], which provides the standard means to control and monitor hardware components. HPI is out of the scope of this thesis and our focus will center on the services defined by AIS.

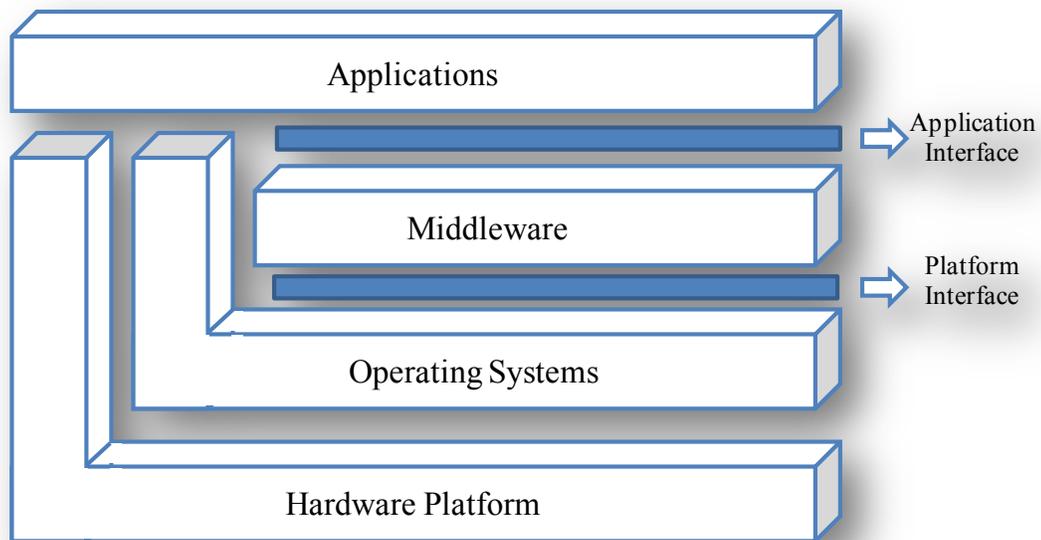


Figure 2-1 The Service Availability Interfaces

AIS is divided into smaller areas with specialized services that are used together with HPI to manage the redundant components of the applications and the underlying hardware.

2.1.3 The Availability Management Framework

From the availability perspective, the Availability Management Framework (AMF) is perhaps the most important part of the AIS middleware. Its role is to manage the availability of the services provided by an application. AMF fulfills this responsibility by managing the redundant components of an application, dynamically shifting a workload of faulty components to the healthy components.

As mentioned earlier, AMF requires a configuration of the application it manages. This configuration consists of several logical entities that abstract out an application components and services. More precisely, an AMF configuration consists of two different sets of elements: AMF entities and AMF entity types.

2.1.3.1 AMF Entities

AMF entities consist of hardware/software resources, aggregations of resources, constructs supporting redundancy mechanisms, services, and deployment elements (cluster information, number of nodes, etc.).

Component

A component represents hardware or software resources capable of supporting the workload of the application services. It is the smallest AMF logical entity on which AMF performs error detection and isolation, recovery and repair [SAF 2010d].

Component Service Instance (CSI)

The Component Service Instance represents the workload that AMF assigns to a component. AMF assigns High-Availability (HA) states of active and standby to components for handling their component service instances depending on whether the component is active (it is providing a service) or standby (used as a backup). 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).

Service Unit (SU)

A Service Unit is a logical entity that aggregates a set of components, combining their individual functionalities into a higher level service. SU is the basic redundancy unit for AMF and can have the HA (High Availability) active state, the HA standby state or no HA state on behalf of a Service Instance (SI).

Service Instance (SI)

The aggregation of components enables the combination of their functionalities to form into 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.

Service Group (SG)

A Service Group aggregates a set of service units that collaborate in a redundant manner in order to protect a set of SIs by means of redundancy. The service group also defines the level of protection applied to the SIs. This is achieved through five different redundancy models defined in AMF specifications [SAF 2010d]. These redundancy models differ on the number of SUs that can be active and standby for the SIs and on how these assignments are distributed among the SUs. The following is the list of the redundancy models defined by AMF:

- 2N Redundancy Model: 2N redundancy model requires two SUs. One SU is active for all the SIs protected by the SG and one is standby for all the SIs.

- N+M Redundancy Model: 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.
- N-Way Redundancy Model: 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 SU while it can be assigned standby to several service units.
- N-Way-Active Redundancy Model: An SG with the N-Way-Active redundancy model has N SUs which are assigned only as active. It has no SU assigned as standby. Furthermore, each of the SIs protected by this SG can be assigned to more than one SU.

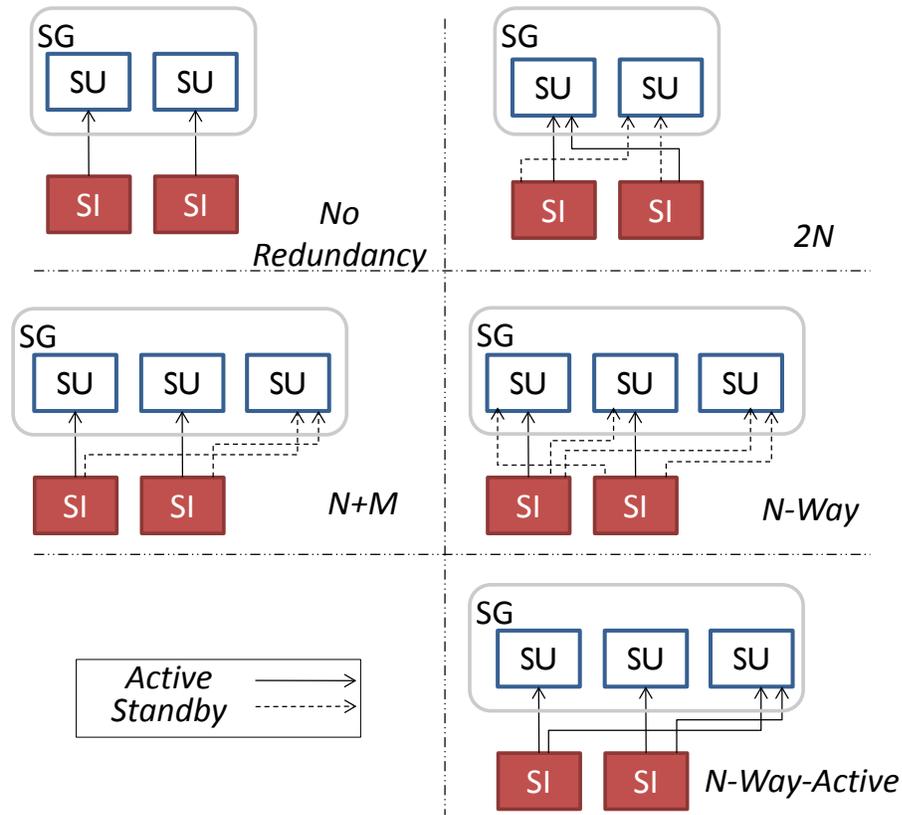


Figure 2-2 Redundancy models defined in the AMF specification

- “No-Redundancy” Redundancy Model: It consists of one or many service units that handle the entire set of SIs protected by the SG in their active state. There are no standby assignments. The difference with the N-Way-Active redundancy model is that in this case each service instance is assigned to at most one service unit and each service unit can protect at most one service instance.

Figure 2-2 summarizes the different redundancy models defined in the AMF specification.

Application

To provide a higher level service, a set of service groups is aggregated into an application. While an application can contain multiple service groups, each service group belongs to only one application.

Node and Cluster

All the aforementioned AMF entities are hosted on AMF Nodes. An AMF node is a logical entity on a cluster node. An AMF Cluster is a set of AMF nodes.

Node Group

Each service group has a list of configured nodes that AMF specification referred to as the Node Group.

2.1.3.2 AMF Entity Types

In addition to the entities, the notion of entity type is introduced in the AMF specification to capture common characteristics shared by all the entities that belong to the same type. In AMF all entities except the deployment entities (i.e., node, nodegroup, and cluster) have a type.

Component Type

Each component is typed and its type represents the particular version of the hardware or software used to build that component. It also specifies the component service types a component can support.

Component Service Type (CSType)

A Component Service Type is the type of services a component provides. It is actually a generalization of similar component service instances that are equivalent from AMF perspective and are thus handled in the same manner.

Service Unit Type (SUType)

Each service unit is typed and its type specifies the component types of the components that belong to the service unit of this type. The service unit type also specifies the maximum number of components of each particular type that this service unit type can contain.

Service Type (SvcType)

A Service Type is the type of services a service unit can provide. It also refers to the component service types that are provided by the components of this service unit. For each component service type, the service type constrains the number of component service instances to handle.

Service Group Type (SGType)

A Service Group Type specifies the list of service unit types that a service group of this type can support. All the service groups of a specific type have the same redundancy model.

Application Type

An Application Type specifies the list of service group types that an application of this type can support.

2.1.3.3 Example of an AMF Configuration

Figure 2-3 shows an example of an AMF configuration. Notice that this simple example does not present AMF configurations in their full complexity, but rather, introduces the reader to the fundamental concepts in these configurations. In this example, a cluster is composed of two nodes (Node1 and Node2). It hosts an application consisting of one SG protecting two SIs (SI1 and SI2) in a 2N redundancy model. The SG consists of two SUs, SU1 and SU2, each being composed of two components.

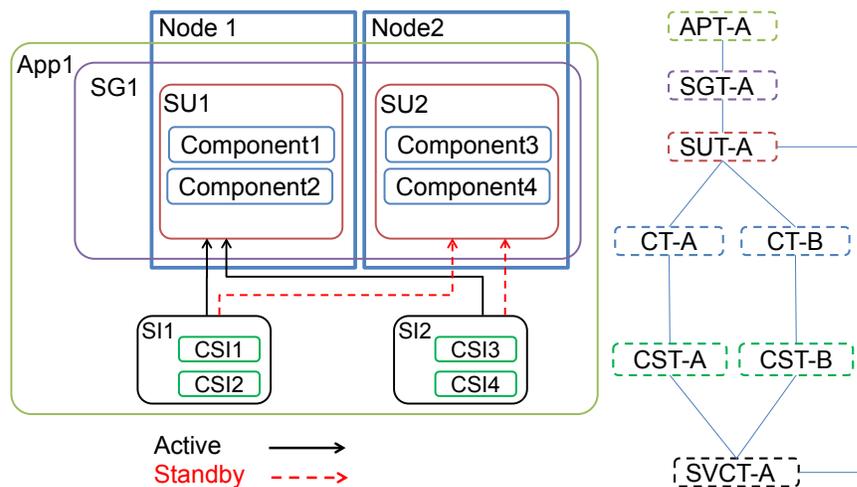


Figure 2-3 An example of an AMF configuration

Although shown in Figure 2-3, 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 runtime. The relationship between the type entities and the entities presented in the configuration are as follows: Component1 and Component3 are from the Component

Type CT-A, while Component2 and Component 4 are from CT-B. Both the SUs are represented by the same SUType called SUT-A. SG1 and App1 are from the type SGT-A and APT-A, respectively. At the service level, both SIs are from the type SVCT-A while the CSIs are from two different types. More specifically, CSI1 and CSI3 are of the type CST-A, while CSI2 and CSI4 are from the type CST-B.

2.1.4 The Entity Types File

In order to design an AMF configuration for a given software system, it is necessary to have a description of the software's components, their capabilities, supporting services, as well as the constraints on any of the parameters and their combination options. This description is provided by the software developer in the form of another SA Forum standard, known as the Entity Types File (ETF) XML schema. Using ETF, software developers can specify the characteristics of their software, capabilities, and limitations in a way that can guide the generation of an AMF configuration. Moreover, ETF elements (referred to as ETF types) describe how an application's components can be combined by providing information regarding their dependencies and compatibility options.

An ETF file must provide at least two types: the Component Types and the Component Service Types (CSTypes). Other entity types such as Service Type (SvcType), Service Unit Type (SUType), Service Group Type (SGType), and the Application Type (AppType) may also be used in order to capture the limitations and constraints of the application. However, they do not have to be provided in ETF.

For instance, Figure 2-4 shows the ETF types that are used to generate the AMF configuration shown in Figure 2-3. The ETF model specifies the Component Types CT-

AA, CT-BB and CT-CC. CT-AA provides CST-AA, while CT-BB provides CST-BB and CT-CC provides CST-CC. CST-AA and CST-BB are grouped in the service type SVCT-AA. CST-BB and CST-CC in the service type SVCT-BB while the service type SVCT-CC aggregates CST-CC. Moreover, CT-AA in providing CST-AA requires CT-BB to provide CST-BB. Finally, there exists an SUType (SUT-AA) aggregating CT-AA and CT-BB that provides SVCT-AA.

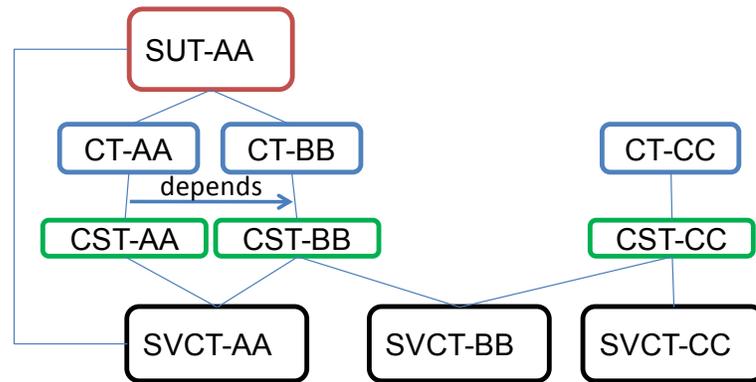


Figure 2-4 An example of ETF model

ETF entity types and AMF entity types describe the same logical entities from two different perspectives. AMF deals with types from a configuration and runtime management point of view, while ETF projects the description of the software from the vendor’s point of view and describes the ways the software could be deployed and its various capabilities and limitations.

2.2 Modeling and UML Profiles

Our proposed approach for defining the framework for AMF configuration management is based on the model-driven paradigm. Moreover, one of the key aspects of our approach is the definition of a domain specific modeling language which captures AMF domain concepts. More specifically, in the proposed solution we have extended the UML

metamodel by means of the UML profiling mechanism [Abouzahra 2005]. By doing so, we aim to take full advantage of UML as being the de facto standard for modeling (e.g. standard tools support interoperability with other OMG standards) and design while having a precise language tailored for AMF concepts and semantics. In this section, we review key concepts that pertain to the development of domain specific modeling languages, and the UML profiling mechanism. We also report on other UML profiles related to our research.

2.2.1 The UML Profiling Mechanism

2.2.1.1 Domain Specific Languages & Domain Specific Modeling Languages

Domain-specific languages (DSLs) are languages tailored to a specific application domain. They are easy to use and provide an extensive level of expressiveness for users [Mernik 2005]. As a matter of fact, domain specific elements are more appropriate for communication with users. In addition, contrary to general purpose languages, DSLs have a limited semantic scope and reduce development challenges substantially. The domain specific modeling (DSM) approach has been introduced in order to utilize DSLs for the modeling and analysis of concepts within certain domains [Kelly 2008]. For this purpose, the concept of domain specific modeling languages (DSML) emerged. Moreover, due to the popularity and extensive advantages of the Unified Modeling Language (UML) [OMG 2007a]—a general purpose language—, UML has been broadly employed by many software practitioners as a DSML [Abouzahra 2005, Felfering 2000].

2.2.1.2 UML Extension Mechanisms

The OMG (Object Management Group) [OMG 2011] defines UML [OMG 2007a] as a visual language for specifying, designing, and documenting the artefacts of a wide variety of systems (e.g. software systems, real-time systems or business process models). In addition to being an extensively accepted standard for object-oriented modeling in the software engineering community, UML is also supported by panoply of existing CASE tools. It is a general purpose modeling language that covers a variety of domains from different points of view and involves different levels of abstraction [Fuentes 2004]. However, there are circumstances in which UML is too general and thus inappropriate for modeling applications within specific domains. In such cases, UML can be extended using one of the following mechanisms [Fuentes 2004]:

- A heavyweight extension mechanism which enlarges the UML metamodel by adding new model elements. This can be achieved by extending the UML metamodel through Meta-Object Facility (MOF) [OMG 2006a], which defines the UML metamodel itself. Some examples of using the heavyweight UML metamodel extension mechanism can be found in [OMG 2003b, Knapp 2003].
- The lightweight extension mechanism, which consists of adding and/or modifying the semantics of UML elements through its metamodel. The newly introduced elements form a UML profile, which is usually a package that contains the new elements and describes how they map to UML metamodel elements [OMG 2002, OMG 2004].

The first approach is more expressive since it enables the definition of a tailor-made language for defining a notation that precisely matches the concepts of the target domain.

However, this approach cannot be supported by most standard commercial tools. On the contrary, using UML profiles provides compatibility with UML modeling tools, though it may result in less accuracy, and the newly introduced elements may not perfectly match domain specific concepts. In fact, choosing between these two approaches is not a straightforward decision. Due to the complexity of the heavyweight mechanism, it seems that, unless there is a real necessity to deviate from the UML metamodel, the advantages of using UML profiles outweigh its restrictions [Fuentes 2004].

2.2.1.3 Creating a UML Profile

Unfortunately, there has been little material 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 being of poor quality [Selic 2007, Lagarde 2007, Lagarde 2008]. To address this issue, Selic describes [Selic 2007] a systematic approach for defining profiles. He proposes a two-step approach which consists of the following:

- Specifying the domain model (or domain metamodel): The domain model specifies the concepts that pertain to the DSL and how these concepts are represented. The output of this phase consists of fundamental language constructs, relationships between domain concepts, constraints imposed by the domain, the concrete syntax or the notation used to render these concepts, and the semantics of each language construct.
- 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 that 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.

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 the most challenging activity in defining a UML profile. For example, since there is no systematic approach for selecting the most suitable meta-classes, the designer may end up with several candidates for a single domain concept. Accordingly, this phase extensively depends on the experience of the profile's designer. Other studies [Lagarde 2007, Lagarde 2008] propose patterns that are based on a few types of relationships that may exist between domain elements and the corresponding meta-classes. However, these guidelines focus on specific scenarios and do not provide a general solution to the mapping problem. In other words, there is no "ready to use" solution that addresses the general issue of selecting the most appropriate UML metaclass for a specific domain element. In this thesis, we carefully selected the UML meta-classes that best fit the AMF concepts through thorough examination of the UML metamodel.

2.2.2 Related UML Profiles

There are several UML profiles (some of them standardized) that model concepts such as components and services, which are also key concepts in AMF. Some of these profiles also target dependability analysis by facilitating the mapping to analytical models such as Petri nets and fault trees. 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 2011] and which represent some concepts that are also found in AMF. These profiles are: SPT [OMG 2003], MARTE [OMG 2009], and the UML profile for QoS&FT [OMG 2008]. There exist also other profiles that are related to the AMF concepts, namely the DAM Profile [Bernardi 2008] 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 OMG profiles mentioned above.

The UML SPT profile [OMG 2003] 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, by reusing SPT, one should define all necessary constructs for AMF configurations and for ETF. However, basing this definition on SPT's abstract syntax may increase the complexity of designing our language by imposing extra constraints unrelated to the AMF domain.

The MARTE profile [OMG 2009], the successor of SPT, defines a package for Non-Functional Properties (NFP) that supports new user-defined NFPs for different specialized domains [OMG 2009]. It also defines a package for the purpose of analysis called the Generic Quantitative Analysis Modeling (GQAM). However, similar to SPT, none of the newly introduced concepts in MARTE are sufficient for modeling and analyzing aspects of service availability. MARTE does not concentrate on availability concepts such as the redundant structures which play a crucial role in highly available systems. In order to reuse MARTE for our domain, one can only use the basic building blocks of MARTE which have been designed for the purpose of capturing quality attributes other than availability. In other words, the building blocks of MARTE enforce constraints related to non-functional attributes other than availability. Consequently, reusing these building blocks does not facilitate the design of AMF configurations, and also generates much more complexity.

The 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 characteristics, focusing on the availability attributes such as mean time to failure. Although there are many availability related attributes introduced in this profile, it does not support the constructs that are necessary for designing highly available systems such as redundancy structures. In order to reuse this profile for the AMF configuration management domain, we still need to build all required constructs and fundamental structures and embed generic concepts introduced by QoS&FT in these structures. In this case, it is necessary to create relationships between the AMF structures and the attributes of this profile. Moreover, the concepts introduced in this profile are

rather too general to be used for AMF. Therefore, we need to further specify constraints in order to make them specific to our domain. By introducing a UML profile, one can define the availability attributes inside the building blocks themselves (instead of making relationships to external entities) and thus, there is no need for any further refinements.

Both the 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 2008]. 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). However, in order to represent these concepts in the AMF configuration domain model, we have introduced two sets of domain entities (ServiceProvider Package and Service Package). Both packages contain several domain entities (e.g. Component Service Instance, Proxy Component, Service Unit, Service Instance, etc.) which cannot be modeled by the DAM profile. Moreover, there is a substantial distinction between the concept of service in DAM and in our domain. 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 the workload to be assigned to service providers at runtime. To bridge the gap between the definition of services in DAM and AMF, we either need to ignore the service part of the DAM profile and completely re-build the service structures, or specify a large number of complex constraints to adapt the existing definition of services to our context. Both cases are practically equivalent to the creation of entirely new structures and concepts.

The HIDDENETS profile [Kövi 2007] was introduced to model software that runs on the HIDDENETS platform. The HIDDENETS 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 2010b], AUTOSAR Profile [OMG 2006b], and MAM-UML [Belloni 2006]. In addition, the HIDDENETS profile is compliant with the AMF specification [SAF 2010d]. HIDDENETS utilizes AMF concepts using the facade design pattern and makes the AMF related concepts transparent to the user. HIDDENETS, however, only relies on AMF related APIs instead of modeling AMF concepts. Also, the objective of HIDDENETS, which consists of addressing a specific set of applications, is different from our goal, which is specifying and analyzing AMF configurations.

The recently published work described in [Szatmári 2008] is probably the work most related to our research stream. The authors of this paper introduced an MDA (Model-Driven Architecture) approach for the automatic generation of SA Forum compliant applications. They have introduced a metamodel based on the AMF specification [SAF 2010d]. Based on the authors' approach, an application is first modeled using their metamodel (Platform Independent Model) and then mapped to an APIs (Platform Specific Model) that represents the implementation of SA Forum services. The work in question concentrates more on application development than on configuration generation. Assuming to have all the required information for the software, the authors ignore the role of the entity types file (ETF) in their framework which is an important part of creating a configuration. In order to establish a modeling framework, they present a UML

profile based on AIS standards. However, the introduced profile seems to have several shortcomings, such as the following:

- The profile does not guarantee valid configurations since constraints on AMF concepts are not captured. This is due to the fact that the authors simply modeled AMF concepts based on a class diagram given in the AMF specification. This diagram, however, does not model the AMF constraints on these concepts. The constraints are captured in other parts of the specification. In our work, a tedious step was dedicated to capturing domain specific constraints and to specifying those constraints using Object Constraint Language (OCL) [OMG 2010a].
- In their profile, the authors have specified stereotypes for runtime entities of which the configuration designer does not have any control at configuration time.
- The authors have mapped all domain concepts to the UML metaclass Component. Considering the fact that we have deployment concepts or service concepts in this domain, mapping all of the domain concepts to the metaclass Component appears to have not been a proper design decision.
- As a general purpose modeling language, UML provides an extensive level of flexibility. Therefore, in order to specify a UML profile, certain constraints are required to restrict the UML metamodel. Similar to domain specific constraints, there are no constraints specified regarding this aspect.

Chapter 3

Modeling Framework- Domain Models

In this chapter, we present the domain model for modeling framework. This modeling framework is defined by extending UML through its profiling mechanism which results in a UML profile for: 1) AMF configurations, 2) Entity Types File, and 3) Configuration Requirements (CR). Therefore, the modeling framework is composed of three UML sub-profiles, namely the AMF, ETF and CR sub-profiles.

The process of creating the profile consists of two phases. The first phase is concerned with specifying the domain model of the profile, which formally describes the concepts of the domain, the relationships among them, as well as the domain specific constraints. The second step consists of mapping the domain model to the UML metamodel by defining a set of stereotypes, tagged values and constraints (see Chapter 4). This phase requires identifying the most appropriate UML concepts, represented as UML metaclasses, which need to be extended to support the domain concepts. The criteria we followed for building the profile consists of:

- 1) ensuring completeness by containing all the elements needed by the domain;
- 2) not contradicting nor violating the UML metamodel;
- 3) reusing metaclasses based on their semantics;

- 4) reusing as many UML relationships between the stereotyped elements as possible;
- 5) constraining the stereotyped elements to behave according to the rules of the domain.

In this chapter we present the domain model of our modeling framework. The next chapter is dedicated to discussing the mapping of the domain model to the UML metamodel. The content of this chapter has been published in [Gherbi 2009, Salehi 2010a, and Salehi 2011b].

3.1 Domain Modeling Process

We developed the domain model of the profile by studying the specifications and through constant interactions with a domain expert. In our domain modeling process we went through several iterations in order to ensure that the concepts of the domain model were captured properly. We have focused on different specifications and resources in order to capture the concepts of our domain model. More specifically, we studied the AMF specification [SAF 2010d] in order to extract the AMF configuration domain model while the ETF domain model is designed by studying the ETF standard XML schema [SAF 2010e]. The 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 3-1 represents the process of specifying the domain model.

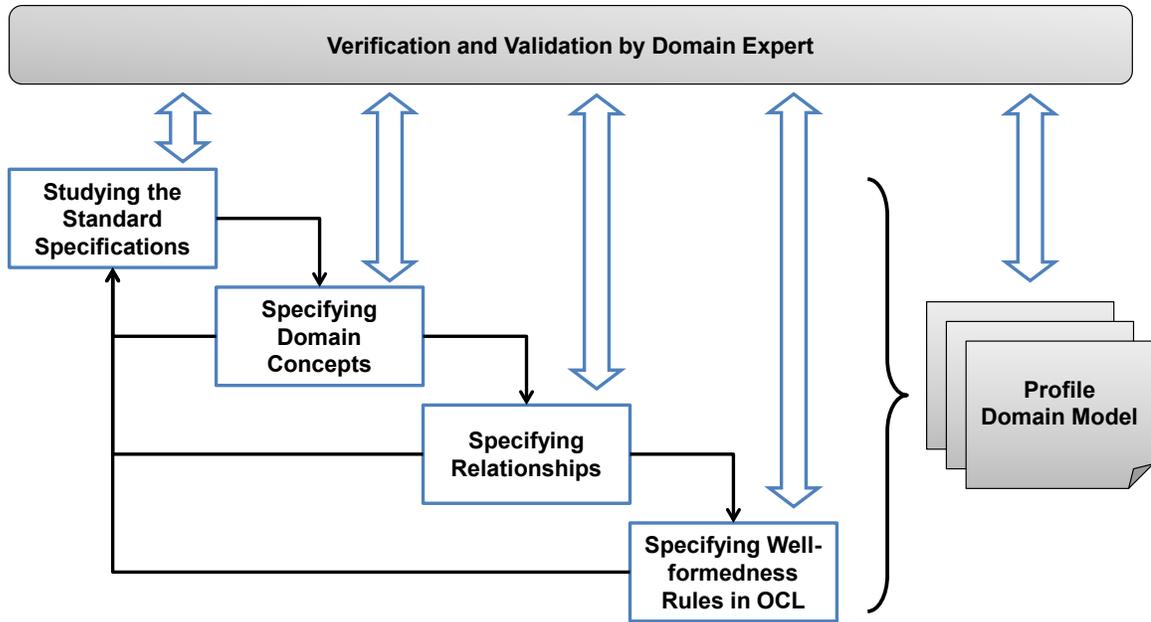


Figure 3-1 Domain Modeling Process

3.2 AMF Domain Model

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

life-cycle of contained components. Finally, the locality criterion distinguishes components that reside within an AMF cluster from the external ones. External components are also proxied to be controlled by AMF. The majority of components managed by AMF are expected to reside within the AMF cluster. The SA-aware components, regardless of the other criteria (containment and proxy-based mediation), are inevitably local. The local components category also includes the non SA-aware components which are either proxied or not proxied.

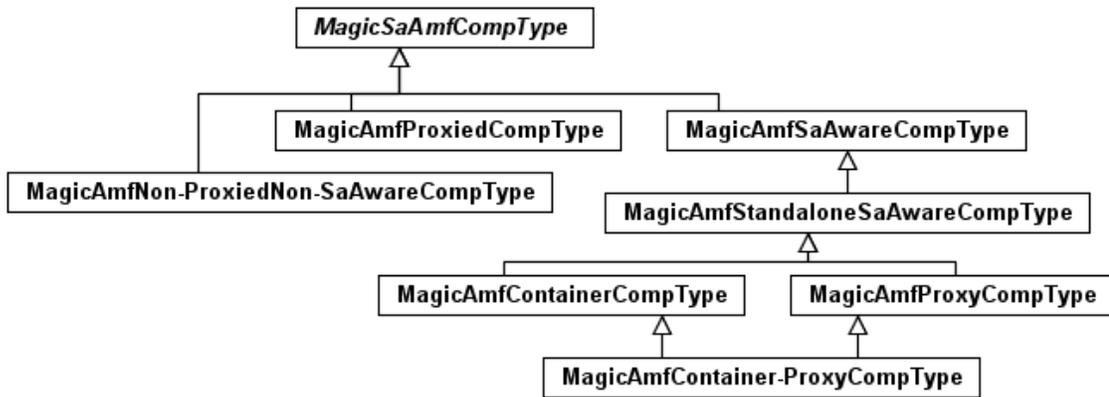


Figure 3-3 AMF Component Type Categories

Unlike the component classification, our classification of the component types does not take into consideration the locality criterion. This is because the component type cannot specify whether its components have to be 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, a standalone

component type is further specialized into a proxy component type and a container component type which are the types of the proxy and container component, respectively.

Figure 3-3 represents the categories of AMF component types.

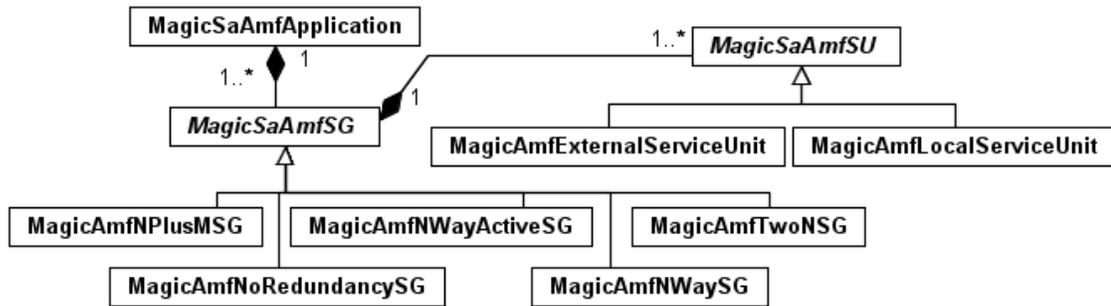


Figure 3-4 Service Unit and Service Group Categories

3.2.2 SU, SG, SI, CSI and their Types

To provide a higher level service, components are grouped into SUs. We distinguish between local and external SUs (see Figure 3-4) based 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 3-4). 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 3-5.

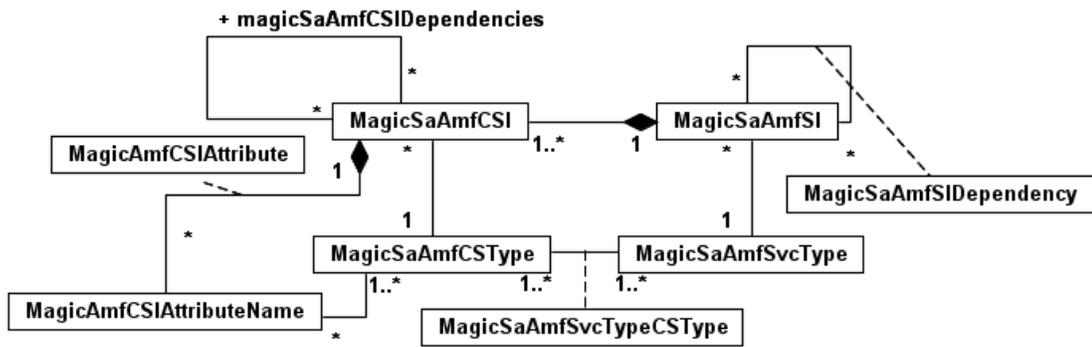


Figure 3-5 Component Service Instance and Service Instance

3.2.3 Deployment Entities

The cluster, the node and the nodegroup represent part of our model for the deployment entities (see Figure 3-6). An AMF cluster is a complete set of AMF nodes in the AMF configuration. A node represents a complete inventory of the SUs and, consequently, the corresponding components that it hosts. A nodegroup represents a set of 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 nodegroup, giving an AMF implementation multiple options for deploying the SU. Moreover, if a failure occurs on a hosting node, for each of the SUs deployed on the faulty node, AMF must select another host node from their configured nodegroup.

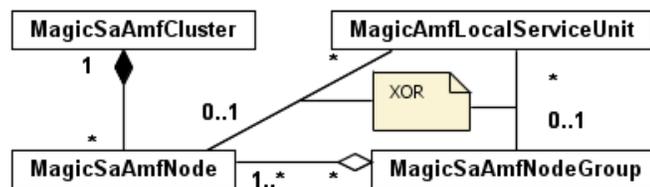


Figure 3-6 AMF Nodes, Node Groups, and Cluster

3.2.4 Well-formedness Rules

We use OCL to describe the constraints on the AMF domain model elements. These constraints govern both the structure and the behaviour of these entities. 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. In the rest of this subsection, we describe each category along with a representative example.

3.2.4.1 Configuration Attributes Well-formedness Rules

As discussed in Chapter 1, one of the main reasons for the complexity of AMF configurations is the large number of configuration attributes and parameters to be considered and the constraints on their values. These constraints form the category addressing the well-formedness rules concerning the configuration attributes. In other words, this category represents the constraints imposed by the AMF domain on the configuration attributes of different domain elements. For instance, among the attributes of the component type element, the *magicSaAmfCtDefDisableRestart* attribute specifies whether the restart recovery action is disabled for components of this component type and the *magicSaAmfCtDefRecoveryOnError* attribute specifies the default recovery action that should be operated by the middleware for the components of this type. Based on the AMF domain, for a certain component type, if the *magicSaAmfCtDefDisableRestart* is configured true, then the attribute *magicSaAmfCtDefRecoveryOnError* must not be set to *SA_AMF_COMPONENT_RESTART* or *SA_AMF_NO_RECOMMENDATION*. This constraint is 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 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.

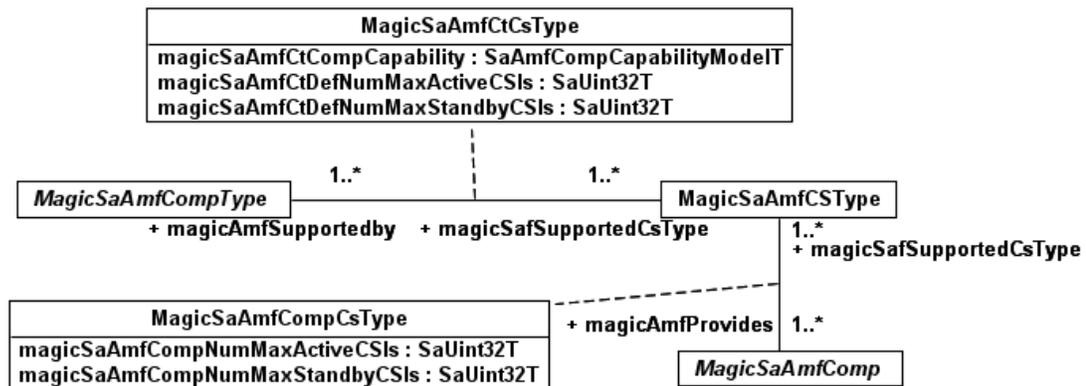


Figure 3-7 Relationship of CStype with component and component type

Figure 3-7 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 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)
```

3.2.4.2 Structural Well-formedness Rules

The elements of 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. We can therefore easily capture this restriction in OCL as follows:

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

3.2.4.3 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. 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 makes the problem complex in most redundancy models. For instance, the problem has combinatorial aspects in N-Way and N-Way-Active redundancy models where the SIs can be assigned to more than one SU simultaneously. 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 model, the necessary and sufficient conditions can be expressed using first-order predicate logic and therefore for these cases the well-formedness rules are specified in OCL. For example the conditions for the case of 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 standby for all service instances protected by the service group.

The OCL constraints specifying the well-formedness rule for the active 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.magicAmfLocalComponentMemberof->iterate(c ; a:integer = 0|
c.MagicSaAmfCompCsType->select (compcst | compcst.
magicSafSupportedCsType = cst)->
asSequence.at(1).magicSaAmfCompNumMaxActiveCSIs+a))))))

and

(self.magicAmfSGGroups->forall(su |
su.oclIsTypeOf(MagicSaAmfExternalServiceUnit))
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.magicAmfExternalComponentMemberof->iterate(c ; a:integer = 0|
c.magicSaAmfCompCsType->select (compcst | compcst.
magicSafSupportedCsType = cst)->
asSequence(1).magicSaAmfCompNumMaxActiveCSIs+a))))))

```

However, for the N+M, the N-Way-Active, and the N-Way redundancy models, the problem is combinatorial and NP-hard [Salehi 2009]. For these cases, the necessary and sufficient conditions are specified in higher order logic (HOL). Due to the fact that OCL is based on first order predicate logic, it is not suitable for expressing these constraints. For overcoming this complexity, we have characterized a special set of SIs, where the necessary and sufficient conditions have been defined and can be checked using 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 Chapter 5.

3.2.5 Challenges

The AMF specification served as our main source for understanding and capturing the concepts of the AMF domain model. This specification defines what a valid AMF configuration is and how it is managed at runtime by a compliant AMF middleware implementation. Therefore, in order to design the AMF domain model, it is necessary to distinguish clearly between configuration time and runtime aspects. This process was not straightforward since often the specification does not provide a clear cut answer as to whether aspects are necessary criteria for configuration or AMF service runtime related requirements. As specification defines relations between the different entities involved in a configuration, there is a temptation to define all of them at configuration time. This is not a valid decision, as some of these relations are defined to allow more flexibility for the AMF middleware at runtime. These runtime relations are based on other configuration time constraints to ensure that the configured application will provide and protect the service independently from the decisions taken by the middleware. Capturing and specifying these configuration time constraints without the related runtime relationships between the entities is not a simple process. Moreover, it is not clear which one of these aspects should be captured in the domain model and to what extent. Indeed, here we are facing the traditional over- vs. under-specification problem. Over-specification occurs when we try to capture some concepts and/or constraints in our domain model which are not configuration time and instead are 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 do not capture configuration time relations. Such misinterpretations could result in a profile that either excludes valid AMF

configurations as a consequence of over-specification or which includes invalid configurations. Close interaction with the domain expert and several iterations allowed us to avoid some pitfalls that would have led to over- or under-specification. For instance, one of the most important AMF requirements 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. The interactions with the domain expert showed that this relationship is not a configuration time relationship and it is only at runtime that an AMF middleware selects and assigns a particular proxy component to a particular proxied component. This association is therefore removed from our model, as it represents a typical case of over-specification, which fixes runtime relationships at configuration time.

3.3 ETF Domain Model

SA Forum standards informally define the specification of the software components by means of XML files called Entity Types File (ETF). ETF as defined in the standard specification is rather ambiguous and informal. Due to the hierarchical representation of XML documents, the relationships between the elements are defined in a uni-directional manner. For instance, CTypes are defined as children of their supporting Component Types. Therefore, in order to find out which Component Types support a certain CType, one should explore all Component Types and find the ones having that CType as one of their children. Moreover, the set of constraints—one of the most important aspects of the domain model—is not complete and a few constraints that are explicitly defined in standard specifications are specified in natural language. Therefore, in order to thoroughly capture the concepts of this domain, we went through constant interactions

with an ETF domain expert. In the rest of this section we present the concepts captured in the ETF domain model.

3.3.1 Basic Service Provider and Service Elements

The basic software entities in ETF are component types which represent the characteristics of the software resources and the various ways they can be configured from the vendor's point of view, such as: 1) the capability of the instances of the software entity in handling the active and/or standby assignments and 2) the compatibility of the instances of component types for the purpose of interacting with instances of other component types. ETF supports the notion of component base type which defines the configuration attributes common to its different versioned component types. We have classified ETF component types according to three different criteria: service availability awareness (SA-awareness for short), containment, and proxy mechanism (see Figure 3-8). The SA-awareness criterion distinguishes the Component Types that implement the AMF APIs and which directly interact with an AMF implementation to manage service availability. The SA-aware Component Types are further specialized into the independent Component Types whose instances can be run on the middleware without any mediation. On the contrary, the contained Component Types do not run directly on an operating system but instead use an intermediate environment. These intermediate environments, like a virtual machine are instances of another category of ETF independent Component Types called container Component Types. Container Component Types are software designed to allow AMF to control the life-cycle of contained Component Types. Proxy Component Types are, however, used to give AMF control over hardware or legacy software, called proxied Component Types.

Finally, the non-proxied non-SA-aware Component Type models the category of Component Types for which the role of the AMF is limited to the management of their life cycle, i.e. instantiation and termination.

The compatibility option which specifies the Component Types capable of collaborating with each other in a redundancy model is captured through the association between “MagicEtfCompType” and “MagicEtfCompBaseType”. We also describe the attributes of software bundles that deliver the Component Types of the model in a class called “MagicEtfSwBundle”.

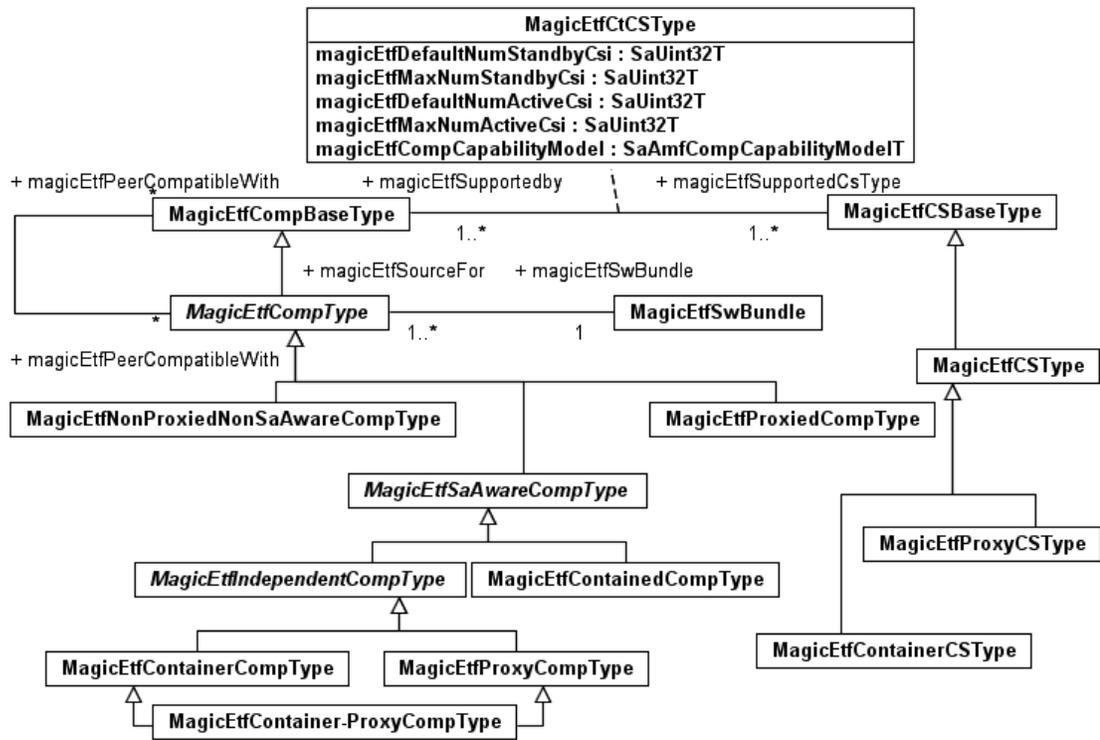


Figure 3-8 Component Type and CStype Categories

ETF CStypes are the description of the workloads that can be supported by the component types. In other words, ETF CStypes model the characteristics of the services which AMF dynamically assigns to components (instances of component types) in terms

of workload. Similar to the component base type, CSBaseType defines the attributes common to its versioned component service types. “MagicEtfCtCSType” association class models the relationship between ETF Component Types and CSTypes. It also specifies the capability of the instances of a given component type in acquiring the workload, i.e. the instances of a certain CSType. More specifically, it describes the capability of software (the maximum that the implementation of software can handle) to act as standby and/or active. In other words, “MagicEtfCtCSType” defines the maximum number of active/standby assignments of the instances of particular CSType to the instances of a specific component type.

ETF CSTypes are further specialized into Proxy and Container CSTypes which are defined to capture the specific proxy and container workloads.

3.3.2 Compound Elements

Compound elements are the elements that represent the combination options of the software elements. More specifically, they specify how software resources can be combined for various purposes, including for the provision of higher level services and the protection of services to ensure service availability. For this purpose, ETF supports different compound elements. The class diagram in Figure 3-9 illustrates part of the domain model which captures the compound elements and their relationships, as well as their connections to the basic elements described in the previous section.

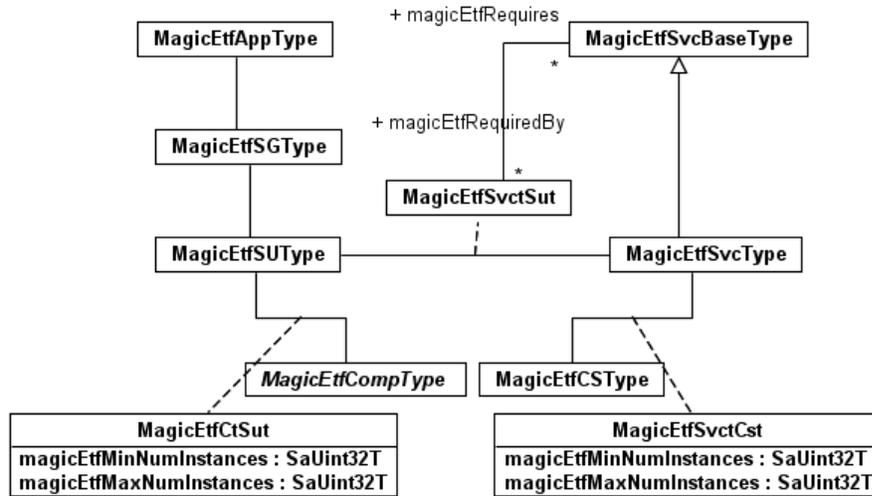


Figure 3-9 Compound elements

ETF SUTypes are the logical software elements that group a set of Component Types. The instances of these Component Types are capable of collaborating with each other to combine their services. Therefore, the software modules associated with the Component Types of a certain SUType are required to implement necessary interfaces in order to collaborate and communicate with each other. Moreover, the limitation on the maximum number of Component Type instances in an instance of a given SUType can be defined by the software vendor. For this purpose, the association class “MagicEtfCtSut”, between “MagicETFSUType” and “MagicETFCompType”, models this constraint through “magicEtfMaxNumInstances” and “magicEtfMinNumInstances” attributes (see Figure 3-9). The set of ETF CSTypes supported by these Component Types also forms another ETF element referred to as Service Type (SvcType). ETF SvcTypes are the description of the workloads that can be supported by SUTypes. Similar to the limitations captured between ETF SUTypes and Component Types, the ETF SvcType may limit the number of the instances of a particular CSType that can exist in an instance of the SvcType. In our domain model, this feature is captured by means of the attributes of the

“MgicEtcSvctCst” association class defined between “MagicEtfSvcType” and “MagicEtcCSType”.

In order to capture the level of service protection provided for the services, another ETF element called SGType is introduced into the domain model. ETF SGType groups a set of SUTypes and specifies the redundancy model supported for the instances of these SUTypes from vendors’ perspective. Therefore, the SGType plays a key role in determining the availability of services. Finally, ETF Application Type defines the set of SGTypes that may be used to build applications, i.e. the instances of the Application Type.

3.3.3 Software Dependency

Software dependency is one of the most important aspects captured in ETF. In the ETF domain we capture the software dependency in two main levels, namely Component Type and SUType levels. There are three different types of Component Type level dependency: CompType/CSType, Proxy/Proxied, and Container/Contained dependencies. CompType/CSType dependency reflects the fact that the provision of a specific service by a certain service provider depends on the provision of another service by a different service provider. In other words, it represents the dependency of a specific Component Type in providing a given CSType on the provision of another CSType by a certain Component Type. This dependency is captured in the ETF domain model through a reflexive association on the “MagicEtfCtCSType” association class (see Figure 3-8). As discussed in Section 3.3.1, components can be of the type Proxied, thus requiring a Proxy that conveys the requests of the AMF middleware. They can also be of the type Contained, requiring a Container capable of managing their life cycles. In the ETF

domain model the Proxy/Proxied dependency is modeled as an association class between the Proxy and Proxied CompType elements. This association class specifies the ProxyCSType provided by the Proxy CompType in order to proxy the Proxied CompType. In this dependency the Proxied CompType relies on the Proxy CompType. Similarly, Container/Contained dependency is modeled as an association class between the Container and Contained CompType elements. The association class specifies the ContainerCSType provided by the Container CompType in order to manage the life cycle of the Contained CompType.

The dependency at the SUType level is specified in the ETF domain model as the dependency of a SUType to an SvcType in providing a given SvcType. In the model the SvcType dependency is defined at the level of a relationship between the “MagicEtfSvcSut” association class and the SvcType class.

3.3.4 Domain Constraints

Specifying constraints is an important step in the definition of a UML profile. In particular, in complex domains class diagrams are absolutely insufficient for expressing all domain specific concepts. In our work, a tedious step was dedicated to capturing domain specific constraints and to specifying those constraints using the Object Constraint Language (OCL). These constraints govern both the structure and the behaviour of these entities. As an example of a constraint definition, let us consider the definition of the following property: *A service unit type that uses contained component types should not use component types of other categories.* This is expressed in OCL in the context of the SUType represented by the class “MagicEtfSUType”. We can, therefore, easily capture this restriction in OCL as follows:

```
context MagicEtfSUType
inv:
self.magicEtfGroups ->
exist (c|c.ocliIsTypeOf (MagicEtfContainedCompType)) implies
self.magicEtfGroups ->
forall (c|c.ocliIsTypeOf (MagicEtfContainedCompType))
```

3.3.5 Challenges

The main challenge in defining the ETF domain model lies in the fact that the main source of information is the standard specifications given as an XML schema. SA Forum standards [SAF 2010b, SAF 2010e] informally define the specification of the software entities by means of XML files. Therefore, the definition of the entities involved in the description of the software is rather ambiguous and informal. For instance, the set of constraints that must be considered between software entities is not complete. Moreover, the few constraints that are explicitly defined in standard specifications are specified in natural language. Recognizing the ambiguous representation of domain concepts in standard specifications, we went through several iterations in order to accurately capture these concepts in ETF domain model. Each iteration consisted of an extensive phase of interactions with the domain expert. At the end of each iteration, the domain model and the document specifying the domain concepts and domain constraints were reviewed by the domain expert and the shortcomings were pointed out and considered in subsequent iterations.

3.4 CR Domain Model

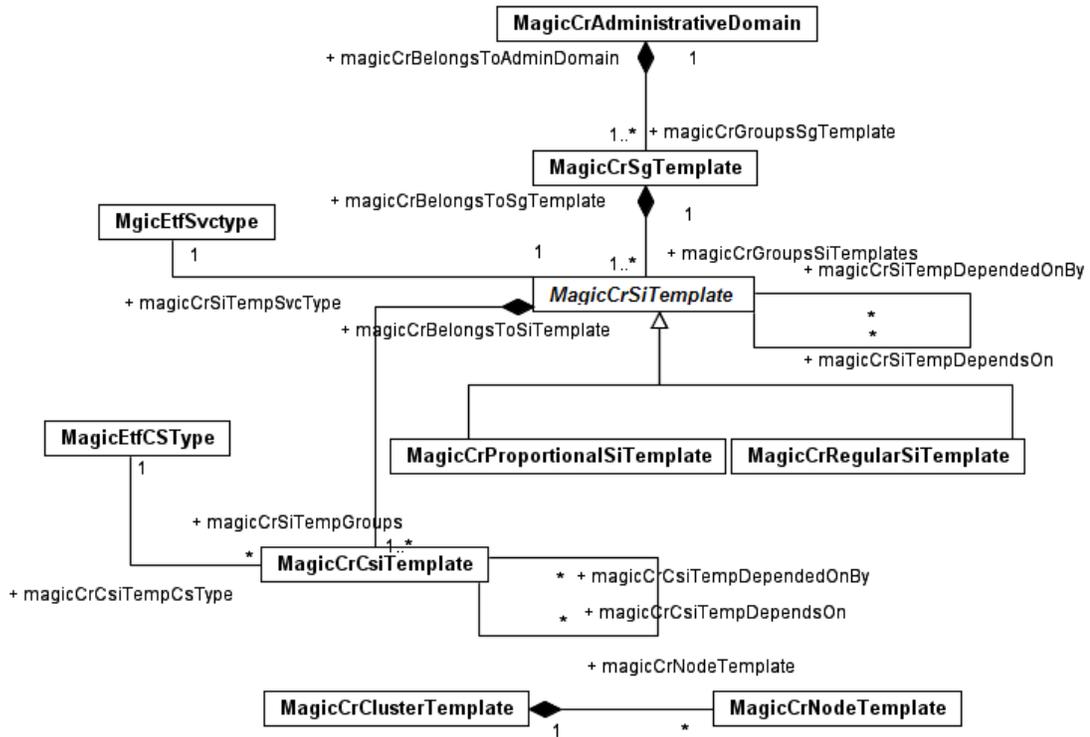


Figure 3-10 Configuration Requirement (CR) domain model

Configuration requirements specify the set of services to be provided by a given software system through the target AMF configuration. More specifically, they define different characteristics of the services such as their types, the number of instances of a certain service type, the relationships between services, and the level of protection expressed in the context of AMF in the form of redundancy models. The configuration requirements model also specifies the requirement for the deployment infrastructure. The specification of the configuration requirements is defined as templates (see Figure 3-10) to help the configuration designer specify common characteristics shared by multiple SGs (through SGTemplates), SIs (using SITemplates) and CSIs (by means of CSITemplates). The CSITemplate defines the information needed to create a set of CSIs. More specifically, it specifies the number of CSIs to be created, the CSType of the created CSIs, and the

relationships between the CSIs. Similarly, SITemplate specifies the SIs with the associated SvcType, the number of SIs to be created, dependencies among SIs, and the set of CSITemplates that constitute the set of CSIs each of the created SIs will contain. The level of protection is one of the most important requirements for the generation of the AMF configurations which is captured through SGTemplate. It specifies the requirements on the SG(s) that will protect the SIs and the sets of SIs that need be protected by this SG(s). The SG template also specifies the redundancy model and the number of SUs in the SG(s) expected to protect the SIs. The number of SUs is divided into two parts: the number of active SUs and the number of standby SUs. The values of the number of active and standby SUs are constrained based on the redundancy model as specified in the AMF specification [SAF 2010d].

In order to group SGTemplates, the CR domain model also introduces the notion of administrative domain. If an SGTemplate belongs to an administrative domain, then all its SITemplates will belong to this administrative domain. The SIs generated from the SITemplates of the same administrative domain can be serviced by the SGs of the same application, and thus at configuration generation time we will associate those SIs only to specific applications defined for the administrative domain.

Finally, The NodeTemplate and ClusterTemplate are used to capture the requirements of the deployment infrastructure, namely the AMF nodes and the AMF cluster. The NodeTemplate specifies the number of nodes and their attributes used to create identical AMF nodes and the ClusterTemplate represents the characteristics of the required AMF cluster.

3.5 Summary

In this chapter we discussed the first phase in defining our modeling framework which concerns specifying the domain model of the UML profile. Our domain modeling process follows an iterative scheme focusing on different specifications and interactions with the domain expert. We discussed the domain model of our profile in terms of three subdomains, namely AMF configurations, Entity Types File, and Configuration Requirements. In each subdomain, we presented the description of the concepts of the domain and the relationships among them, as well as the domain specific constraints.

We also discussed the main challenges we faced in this process and which stem mainly from the informality and incompleteness of the standard specifications for describing ETF concepts and from the fact that the AMF standard specification simultaneously defines what a valid AMF configuration is and specifies the expected behaviour from an AMF service implementation.

The domain modelling process has resulted in three technical reports used in the second phase of the definition of our modeling framework. In the next chapter we present this second phase which consists of mapping the domain model to the UML metamodel by defining a set of stereotypes, tagged definitions, and constraints.

Chapter 4

Modeling Framework- Mapping to UML Metamodel

Once the domain model is completed, the second major step is to map the domain concepts to the UML metamodel. For this purpose, one needs to proceed stepwise through the full set of domain concepts (specified as classes in the domain model) and identify the most appropriate UML base concepts for each of them. 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 introduced stereotypes and the UML metaclass from which each stereotype is derived. It is important to mention that, since UML 2.0 supports inheritance relationships between stereotypes, not all domain concepts need to be directly derived from the corresponding UML metaclasses. Some of them will be derived from the newly created stereotypes. Figure 4-1 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. Following this process, we have carefully 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. Once the stereotypes have been defined, specifying the tagged definitions is the next step in the process of building the concrete syntax of our language. Tagged definitions represent properties of these stereotypes which are not included in UML.

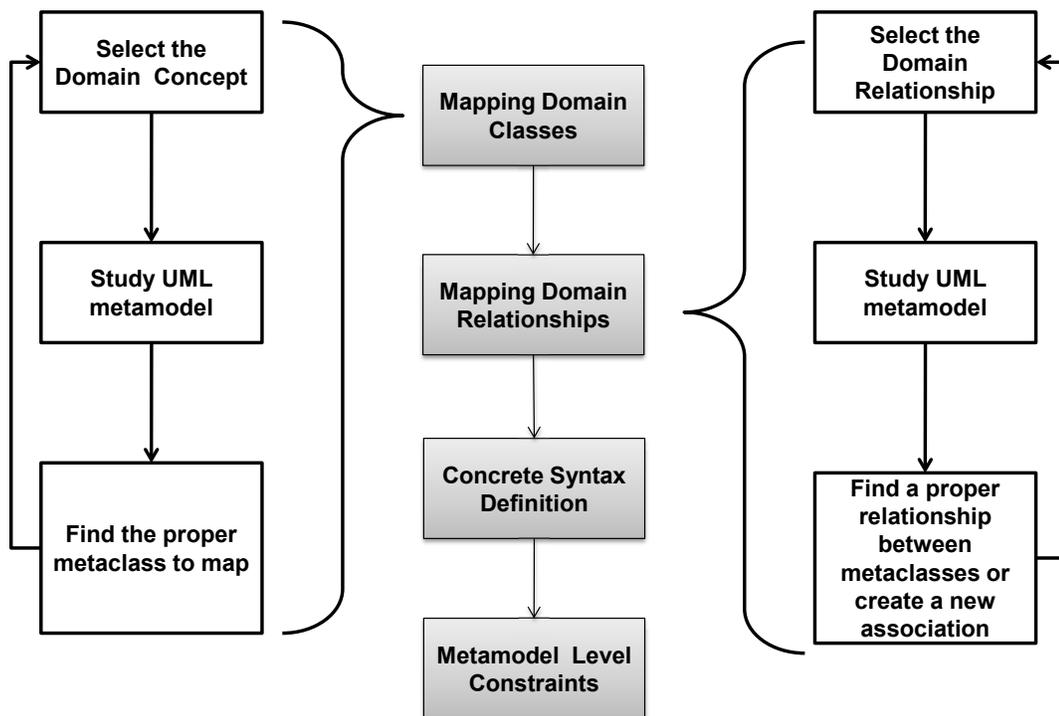


Figure 4-1 The process of mapping to the UML metamodel and concrete syntax definition

Due to the large number of tagged definitions, we present their specifications in Appendix I. The remainder of this chapter is dedicated to presenting the steps of mapping our domain model to the UML metamodel in detail. The content of this chapter has been published in [Salehi 2010a and Salehi 2010b].

4.1 Mapping Domain Model Concepts to UML Metaclasses

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

- The candidate metaclass appears semantically to be appropriate: in this case it is always beneficial to look at the metaclasses inherited from the candidate metaclass. In other words, since the inherited metaclasses specify more features, we may find them semantically more accurate for aligning with the description of the domain concept.
- 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 has fewer features.

These guidelines highly support the semantic alignment of the domain concepts with respect to the UML metamodel. Following this process, we have identified the stereotypes that fit AMF concepts. We present the stereotypes in the next subsections. For each stereotype we discuss the rationale behind the selection of the UML metaclass in question.

4.1.1 AMF Component

The component in AMF represents the encapsulation of the functionality of the 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 a UML component defining a new stereotype called <<MagicSaAmfComponent>>. Similarly, a stereotype is defined for each component category and is indirectly mapped (through inheritance relationships between stereotypes) to the Component metaclass.

4.1.2 AMF Service Unit (SU)

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 a 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>>.

4.1.3 AMF Service Group (SG)

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.

4.1.4 AMF 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>>).

4.1.5 AMF 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 runtime 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>>).

4.1.6 AMF 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 defined 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.

4.1.7 AMF 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.

4.1.8 AMF Cluster and AMF NodeGroup

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.

4.1.9 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.

Table 4-1 represents the summary of the stereotypes defined for AMF entities and entity types as well as the graphical syntax of our language for each stereotype.

Table 4-1 The summary of the stereotypes defined for AMF entities and entity types

Stereotype	Generalization	Notation
<<MagicSaAmfCompGlobalAttributes>>	metaclass Class	
<<SaAmfCompBaseType>>	metaclass Class	BT 
<<MagicSaAmfCompType >>	<<SaAmfCompBaseType>>	
<<MagicAmfSaAwareCompType>>	<<MagicSaAmfCompType>>	T 
<<MagicAmfStandaloneSaAwareCompType >>	<<MagicAmfSaAwareCompType>>	T 
<<MagicAmfProxyCompType>>	<<MagicAmfStandaloneSaAwareCompType>>	
<<MagicAmfContainerCompType>>	<<MagicAmfStandaloneSaAwareCompType>>	T 
<<MagicAmfContainer-ProxyCompType>>	<<MagicAmfProxyCompType>> <<MagicAmfContainerCompType>>	T 
<<MagicAmfProxiedCompType>>	<< MagicSaAmfCompType>>	
<<MagicAmfNon-ProxiedNon-SaAwareCompType>>	<< MagicSaAmfCompType>>	T 
<<MagicSaAmfHealthcheckType>>	metaclass Class	

<<SaAmfSUBaseType>>	metaclass Class	BT 
<<MagicSaAmfSUType>>	<<SaAmfSUBaseType>>	
<<MagicAmfLocalSUType>>	<<MagicSaAmfSUType>>	T 
<<MagicAmfExternalSUType>>	<<MagicSaAmfSUType>>	T 
<<SaAmfSGBaseType>>	metaclass Class	BT 
<<MagicSaAmfSGType>>	<<SaAmfSGBaseType>>	T 
<<SaAmfAppBaseType>>	metaclass Class	
<<MagicAmfAppType >>	<<SaAmfAppBaseType>>	
<<SaAmfCSBaseType>>	metaclass Class	BT 
<<MagicSaAmfCSType>>	<<SaAmfCSBaseType>>	T 
<<SaAmfSvcBaseType>>	metaclass Class	BT 
<<MagicSaAmfSvcType>>	<<SaAmfSvcBaseType>>	T 

<<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	

<<MagicSaAmfSU>>	metaclass Component	
<<MagicAmfLocalServiceUnit>>	<<MagicSaAmfSU>>	
<<MagicAmfExternalServiceUnit>>	<<MagicSaAmfSU>>	
<<MagicSaAmfSG>>	metaclass Component	
<<MagicAmfTwoNSG>>	<<MagicSaAmfSG>>	
<<MagicAmfNPlusMSG>>	<<MagicSaAmfSG>>	
<<MagicAmfNWaySG>>	<<MagicSaAmfSG>>	
<<MagicAmfNWayActiveSG>>	<<MagicSaAmfSG>>	
<<MagicAmfNoRedundancySG>>	<<MagicSaAmfSG>>	
<<MagicSaAmfApplication>>	metaclass Component	
<<MagicSaAmfCSI>>	metaclass Class	
<<MagicSaAmfSI>>	metaclass Class	

<<MagicAmfCSIAtributeName>>	metaclass Class	
<<MagicSaAmfNode>>	metaclass Node	
<<MagicSaAmfNodeGroup>>	metaclass Package	
<<MagicSaAmfCluster>>	metaclass Package	

4.1.10 ETF Types

ETF types describe the characteristics and features of the software entities from the vendor's point of view. These characteristics mainly focus on the aspects of the software which are important for the generation of the AMF configuration. In the process of configuration generation the AMF entity types are created based on ETF types. For instance, ETF defines ranges for some attribute values and consequently, the values of the corresponding AMF type must be between these ranges.

As a result, ETF types act as metatypes for AMF types and, thus, are the generalization of similar AMF types. In UML, the metaclass Class describes a set of objects that share the same specifications of features, constraints, and semantics [OMG 2007b], we have therefore used it as a base class for all ETF types.

Table 4-2 represents the summary of the stereotypes defined for ETF types as well as the graphical notation of our language for each stereotype.

Table 4-2 The summary of the stereotypes defined for ETF types

Stereotype	Generalization	Notation
<<MagicEtfCompBaseType>>	metaclass Class	
<< MagicEtfCompType>>	<< MagicEtfCompBaseType>>	
<< MagicEtfSaAwareCompType>>	<< MagicEtfCompType>>	
<<MagicEtfNonProxiedNonSaAwareCompType >>	<< MagicEtfCompType>>	
<< MagicEtfProxiedCompType>>	<< MagicEtfCompType>>	
<< MagicEtfContainedCompType>>	<< MagicEtfSaAwareCompType>>	
<< MagicEtfIndependentCompType>>	<< MagicEtfSaAwareCompType>>	
<< MagicEtfContainerCompType>>	<< MagicEtfIndependentCompType>>	
<< MagicEtfProxyCompType>>	<< MagicEtfIndependentCompType>>	
<< MagicEtfStandaloneCompType>>	MagicEtfIndependentCompType	
<< MagicEtfContainer-ProxyCompType>>	MagicEtfProxyCompType MagicEtfContainerCompType	
<< MagicEtfSUBaseType>>	metaclass Class	
<< MagicEtfSUType>>	<< MagicEtfSUBaseType>>	
<< MagicEtfSGBaseType>>	metaclass Class	
<< MagicEtfSGType>>	<< MagicEtfSGBaseType>>	
<< MagicEtfAppBaseType>>	metaclass Class	

<< MagicEtfAppType>>	<< MagicEtfAppBaseType>>	
<< MagicEtfSwBundle>>	metaclass Class	
<< MagicEtfUpgradeAwarenessAttributes>>	metaclass Class	
<< MagicEtfHealthcheck>>	metaclass Class	
<< MagicEtfSvcBaseType>>	metaclass Class	
<< MagicEtfSvcType>>	<< MagicEtfSvcBaseType>>	
<< MagicEtfCSBaseType>>	metaclass Class	
<< MagicEtfCSType>>	metaclass Class	
<< MagicEtfContainerCSType>>	<< MagicEtfCSType>>	
<< MagicEtfProxyCSType>>	<< MagicEtfCSType>>	
<< MagicEtfCstAttribute>>	metaclass Class	

4.1.11 CR Elements

Configuration requirement elements represent the description of the configuration and their structure. CR profile is used in the configuration management framework and Table 4-3 presents the summary of the stereotypes of the CR profile.

Table 4-3 The summary of the stereotypes defined for CR elements

Stereotype	Generalization	Notation
<<MagicCrAdministrativeDomain>>	metaclass Class	
<<MagicCrSgTemplate>>	metaclass Class	
<<MagicCrSiTemplate>>	metaclass Class	
<<MagicCrRegularSiTemplate>>	<<MagicCrSiTemplate>>	
<<MagicCrProportionalSiTemplate>>	<<MagicCrSiTemplate>>	
<< MagicCrCsiTemplate>>	metaclass Class	
<< MagicCrClusterTempalate>>	metaclass Class	
<< MagicCrNodeTemplate>>	metaclass Class	

4.2 Mapping the Domain Relationships to the UML Metamodel

We distinguish different categories of relationships between domain concepts:

- AMF domain:
 - 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 relationship which is used for deployment purposes (e.g. between a service unit and a node or between a service group and a node group).
- Member node: represents the relationship which is used between a node and a nodegroup or cluster.
- Contain: represents the relationship between container components and CSI
- Proxy: represents the relationship between proxy components and CSI
- ETF domain:
 - Provide: This relationship is used between service provider ETF types and service ETF types and represents the capability of providing services (e.g. ETF SUType and ETF SvcType).
 - Group: It represents the relationship which is used between grouping and grouped elements (e.g. the relationship between an ETF SUType and its enclosing ETF Component Types).
 - Depend: It represents the dependency relationship which is used between a sponsor and its dependent elements.

- Contain: It represents the relationship which is used between an ETF Container Component Type and its ETF Contained Component Types.
 - Proxy: It represents the relationship which is used between an ETF Proxy Component Type and its ETF Proxied Component Types.
- CR domain
 - Group: It represents the relationship which is used between grouping and grouped elements (e.g. the relationship between a SITemplate and its enclosing CSITemplates).
 - Depend: It represents the dependency relationship which is used between a sponsor and its dependent elements.
 - Type of Service: It refers to the service type needed to be provided in order to satisfy the requirements of CR templates (between ETF SvcType and SITemplate or between ETF CSType and CSITemplate)

A careful selection 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 association has been stereotyped accordingly and mapped to either Association, AssociationClass, or Dependency.

For example, both `<<MagicSaAmfSI>>` and `<<MagicSaAmfCSI>>` stereotypes are mapped to the UML 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 4-2, we defined the stereotype `<<groups>>` to capture the relationship and map it to metaclass `Association`.

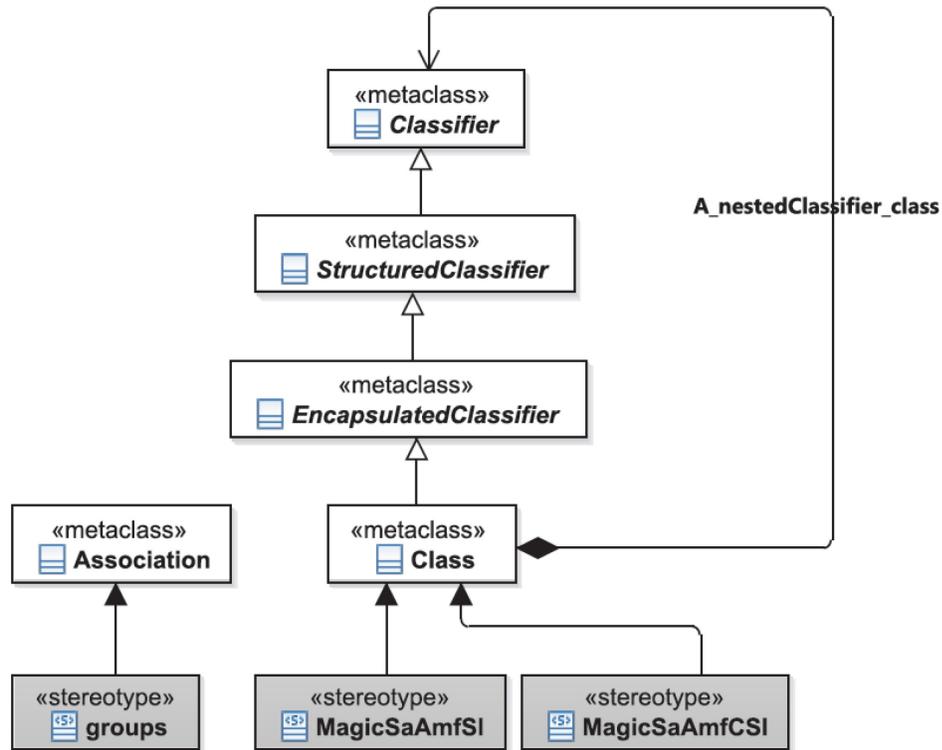


Figure 4-2 Relationship between AMF SI and AMF CSI

Table 4-4 shows a summary of the stereotypes defined for the relationships, their base metaclasses, the relationship reused from the UML metamodel, and the domain.

Table 4-4 Summary of Stereotypes Related to the Relationships between Domain Concepts

Stereotype	UML metaclass	Reused relationship from UML metamodel	Domain
<<groups>>	Association	nestedClassifier relationship between Class and Classifier packagedElement relationship between Component and Packageable Element	AMF ETF CR
<<protect>>	Association	nestedClassifier relationship between Class and Classifier	AMF
<<provide>>	Association	nestedClassifier relationship between Class and Classifier	AMF ETF
<<type>>	Association	superClass relationship between Component and Class Reflective superClass relationship on Class	AMF
<<membertype>>	Dependency	packagedElement relationship between Packageable Element and Package	AMF
<<deploy>>	Dependency	packagedElement relationship between Packageable Element and Package	AMF
<<contain>>	Association	nestedClassifier relationship between Class and Classifier	AMF ETF
<<proxy>>	Association	nestedClassifier relationship between Class and Classifier	AMF ETF
<<typeservice>>	Association	nestedClassifier relationship between Class and Classifier	CR
<<MagicSaAmfSvcTypeCSType>>	Association Class	nestedClassifier relationship between Class and Classifier	AMF
<<MagicSaAmfSvcTypeCSType>>	Association Class	packagedElement relationship between Component and Packageable Element.	AMF

<<MagicSaAmfCtCSType>>	Association Class	nestedClassifier relationship between Class and Classifier	AMF
<<MagicSaAmfCompCsType>>	Association Class	nestedClassifier relationship between Class and Classifier	AMF
<<MagicSaAmfSIDependency>>	Association Class	nestedClassifier relationship between Class and Classifier inherited by AssociationClass	AMF
<<MagicEtfCtCSType>>	Association Class	nestedClassifier relationship between Class and Classifier	ETF
<<MagicEtfSvctSut>>	Association Class	nestedClassifier relationship between Class and Classifier	ETF
<<MagicEtfContainerContained>>	Association Class	nestedClassifier relationship between Class and Classifier	ETF
<<MagicEtfCtSut>>	Association Class	nestedClassifier relationship between Class and Classifier	ETF
<<MagicEtfProxyProxied>>	Association Class	nestedClassifier relationship between Class and Classifier	ETF
<<MagicEtfSvctCst>>	Association Class	nestedClassifier relationship between Class and Classifier	ETF

4.3 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. Since we did not need to specify any constraints on the metamodel attributes, the set of specified constraints were

grouped into two different categories: Constraints on relationships and Constraints on model elements.

4.3.1 Constraints on Relationships

This type of constraints restricts the use of UML relations to the AMF domain. 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. As a result, 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))
```

4.3.2 Constraints on Metaclasses

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()
```

4.4 Challenges

After the analysis of the domain and the design of the domain model, the first issue we faced was how to define our profiles. Although a UML profile may result in a less precise language than a MOF-based language, we avoided a MOF-based solution as this suffers from a lack of tool support. The advantages of a UML profile seem to far outweigh its drawbacks. The second issue involved deciding whether to extend existing profiles or to create a new one. Because of the characteristics of our domain and the fact that the required additional complexity does not justify the very few benefits of a possible extension, we decided to design a new UML profile instead of reusing another profile and adapting it to AMF.

Another challenge that we encountered was in identifying the most appropriate UML metaclasses to extend in order to support domain concepts. We defined some guidelines

for both mapping domain concepts and mapping domain relationships and used them in the mapping 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 statements that require access to stereotyped elements or tagged definitions, RSA implements additional APIs such as *getAppliedSubstereotypes()*, *isStereotypeApplied()*, and *getValue()*. The main issue with these APIs is that they are not compliant with the standard OCL specification and therefore, standard OCL constraints cannot directly be implemented in RSA. Considering the fact that almost all of the constraints in 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.

4.5 Summary

In this chapter we discussed the second step in creating our UML profile which consists of mapping the domain model to the UML metamodel. In this phase we went through three main steps: mapping domain concepts to the UML metamodel, mapping the domain relationships to the UML metamodel, and specifying metamodel level constraints. In the first step the most suitable metaclass was selected for each domain concept by considering the semantic alignment of the domain concepts with UML metaclasses. During the mapping of the domain relationships, in addition to considering the semantic alignment we have also focused on reusing as many UML relationships between the stereotyped elements as possible. This was achieved through a careful selection of

metaclasses for our domain concept from the previous step and resulted in the decreased complexity of the process of defining the profile and in the improved quality of the profile. Finally, we put some restrictions on UML itself by specifying metamodel level constraints in order to guarantee the consistency of the profile with the semantics of the domain model.

We have invested a great deal of effort in improving the quality of our profile by specifying a process for profile definition. In addition, our work has undergone an intensive and effective review process with the domain expert. The applicability and usefulness of the profile will be evaluated empirically in the coming years. This profile serves as the modeling framework for our approaches for model-based configuration generation and the validation of third-party AMF configurations. Both of these approaches either use certain parts of our profile or take advantage of the entire profile. Since our modeling framework is compliant with the UML metamodel, we can transform the configurations into other UML-based analytical models for the evaluation of their availability and other non-functional characteristics.

Chapter 5

AMF Configuration Validation

One of the most important benefits of the model-driven paradigm is the possibility of generating valid artefacts through automated transformations (AMF configurations in our case). However, AMF configurations can also be designed manually by third parties. Considering all the constraints that have to be taken into account and the complexity of the design process, such configurations have to be validated before they can be used by the AMF middleware. These configurations should be:

- Syntactically complete, valid, and consistent with respect to the standard specification of the AMF middleware,
- Semantically aligned with the protection level expressed through characteristics of SGs and the features of the set of SIs configured to be protected by these SGs.

The content of this chapter has been published in [Salehi 2009 and Salehi 2011a].

5.1 Syntactical Validation of AMF Configurations

Having a modeling framework based on the UML, the process of checking the consistency of the model is rather straightforward and is carried out by well-known technologies supporting the UML metamodel. We have used RSA [IBM 2011] to build the AMF profile and the Eclipse EMF [Eclipse 2010b] UML importer to build the Ecore model. The validation process, as shown in Figure 5-1, includes a mapping of an AMF

configuration — provided by the user as an IMM XML [SAF 2010d] file, which is the standard carrier for AMF configurations — to an instance of the AMF profile —presented in this thesis—, as well as a validation of the configuration performed syntactically and with respect to the OCL constraints.

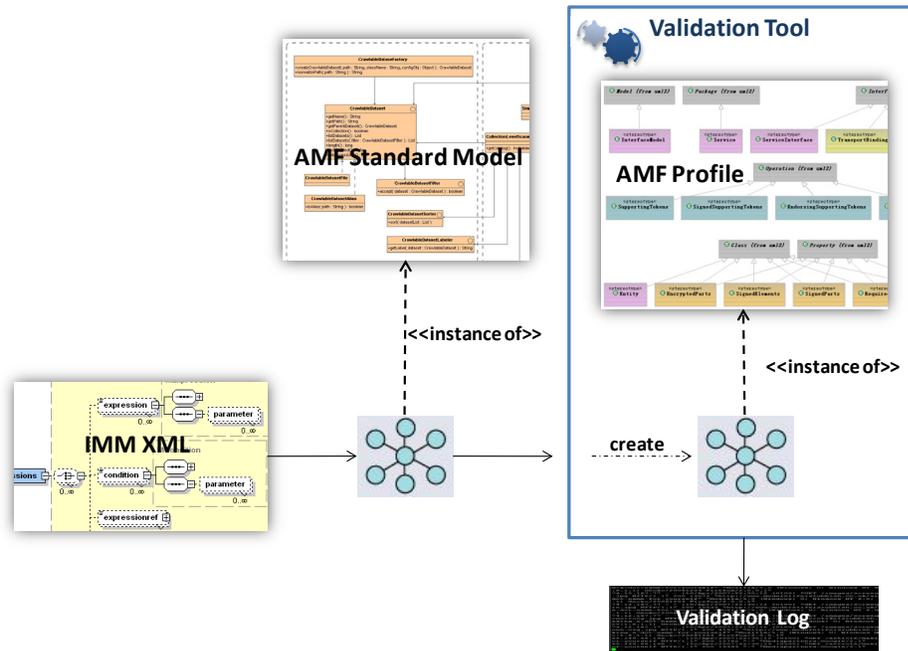


Figure 5-1 Architecture of Validation Tool

5.2 Semantic Validation of AMF Configurations

One of the most important objectives in the semantic validation of AMF configurations is whether a given AMF configuration provides the level of protection it claims or not. In other words, a configuration is semantically valid if and only if it is capable of providing and protecting the services as required and according to the specified redundancy model. Ensuring this requires the exploration of all possible SI-SU assignments and, in some cases, different combinations of SIs; a complex procedure in most redundancy models

defined in the AMF domain. In this section we explore the problem of SI-Protection at configuration time.

5.2.1 Definitions and Notations

Provided services from the provider perspective, or requested services from the requester perspective, can be defined in terms of component service types (CSTypes) and the number of CSIs of each CStype provided or requested, respectively. Therefore, a service group in an AMF configuration can be seen as a set of n SUs denoted by $SUList = \{SU_1, \dots, SU_n\}$. Each SU combines a group of components capable of supporting different CSTypes (i.e. capable of providing the CSIs of those CSTypes) in both active and standby fashion. Let k denote the total number of CSTypes supported by the SUs in a given configuration. Consequently, the provided active capacity list for $SU_i \in SUList$ is defined as $SU_i^{act} = \langle ac_1^i, \dots, ac_k^i \rangle$ and the provided standby capacity list for SU_i is described as $SU_i^{stb} = \langle sc_1^i, \dots, sc_k^i \rangle$. ac_t^i and sc_t^i are non-negative integers representing the capacity of the SU in supporting CSIs from the CStype t .

The n SUs in the $SUList$ need to protect a given sequence of m SIs, denoted by $SIList = \{SI_1, \dots, SI_m\}$. Similar to the provided capacity list of SUs, for each $SI_j \in SIList$, the required capacity list can be defined by two ordered sets $SI_j^{act} = \langle ar_1^j, \dots, ar_k^j \rangle$ and $SI_j^{stb} = \langle sr_1^j, \dots, sr_k^j \rangle$ determining the required capacity of the SI_j for each CStype. In the rest of this section, whenever we use $SU_i = \langle c_1^i, \dots, c_k^i \rangle$ or $SI_j = \langle r_1^j, \dots, r_k^j \rangle$ it implies that the calculation or equation is valid for both active and standby part. Calculating the capacity list of the set of SUs or SIs which is being used through this section is defined in Equation 5-1. This equation defines the summation between two capacity list, but applies

for n ($n > 2$) lists of capacities, where the summation of the first $n-1$ lists is added with the capacity list n , in a recursive manner.

Equation 5-1 Adding capacity lists

$A, B : \text{CapacityList}$ $\text{Let } A = \langle a_1, \dots, a_k \rangle, B = \langle b_1, \dots, b_k \rangle$ $A + B = \langle a_1 + b_1, \dots, a_k + b_k \rangle;$
--

We can assign an SI (SI_j) to an SU (SU_i) in active mode when $SU_i^{act} \geq SI_j^{act}$ and in standby mode when $SU_i^{stb} \geq SI_j^{stb}$ (see Equation 5-2). In other words, SI_j can be assigned to SU_i if and only if the remaining capacity of SU_i for all CSTypes is not less than the capacity required by SI_j . It is important to note that SIs are units of assignment and are indivisible. We also define the division between capacities as given formally in Equation 5-3.

Equation 5-2 Comparison of capacities

$\text{Let } SU_i^{act} = \langle ac_1^i, \dots, ac_k^i \rangle, SI_j^{act} = \langle ar_1^j, \dots, ar_k^j \rangle;$ $SU_i^{act} \geq SI_j^{act} \text{ iff } \forall (1 \leq l \leq k) : ac_l^i \geq ar_l^j$
$\text{Let } SU_i^{stb} = \langle sc_1^i, \dots, sc_k^i \rangle, SI_j^{stb} = \langle sr_1^j, \dots, sr_k^j \rangle;$ $SU_i^{stb} \geq SI_j^{stb} \text{ iff } \forall (1 \leq l \leq k) : sc_l^i \geq sr_l^j$

Equation 5-3 Division between capacities

$A, B : \text{CapacityList}$ $\text{Let } A = \langle a_1, \dots, a_k \rangle, B = \langle b_1, \dots, b_k \rangle$ $A \text{ div } B = \langle \lfloor a_1 / b_1 \rfloor, \dots, \lfloor a_k / b_k \rfloor \rangle;$

In an AMF configuration the assignment of the SIs to the SUs can be defined through the mathematical relations. Equation 5-4 describes the relations capturing the active and standby assignments between *SUList* and *SIList*.

Equation 5-4 Active and Standby relation between a set of SUs and a set of SIs

$$\begin{array}{l} \boxed{SUList \times SIList : ActiveAssignment \subseteq SUList \times SIList} \\ \boxed{SUList \times SIList : StdbyAssignment \subseteq SUList \times SIList} \end{array}$$

In Equation 5-5 we present the mathematical definition of the operators defined for active/standby relation throughout this section. The total active capacity required from an SU *su* in a given SU-SI assignment *A* is denoted by *RequiredActiveCapacityFrom(A,su)* and is defined by the summation of all the required active capacities of the SIs associated to *su* through assignment *A*. Similarly, the total standby capacity required from an SU *su* in a given SU-SI assignment *A* is denoted by *RequiredStandbyCapacityFrom(A,su)* and is defined by the summation of all the required standby capacities of SIs associated to *su* through assignment *A*.

Equation 5-5 Operators for active/standby relation

$$\begin{aligned}
 \text{Range}(A : \text{Assignment}) &= \{y : SI \mid (x, y) \in A\}; \\
 \text{Domain}(A : \text{Assignment}) &= \{x : SU \mid (x, y) \in A\}; \\
 \text{ElementRange}(A : \text{Assignment}, su : SU) &= \{y : SI \mid (x, y) \in A \wedge x = su\}; \\
 \text{ElementDomain}(A : \text{Assignment}, si : SI) &= \{x : SU \mid (x, y) \in A \wedge y = si\}; \\
 \text{RequiredActiveCapacityfrom}(A : \text{ActiveAssignment}, su : SU) &= \\
 \sum_{j=1}^{|\text{ElementRange}(A, su)|} (SI_j^{act}) \quad \text{where} \quad SI_j \in \text{ElementRange}(A, su); \\
 \text{RequiredStandbyCapacityfrom}(A : \text{StandbyAssignment}, su : SU) &= \\
 \sum_{j=1}^{|\text{ElementRange}(A, su)|} (SI_j^{stb}) \quad \text{where} \quad SI_j \in \text{ElementRange}(A, su);
 \end{aligned}$$

Before starting with the redundancy models, we also remind the reader that the AMF specification [SAF 2010d] requires that any SU in an SG must be able to protect any of the SIs protected by the SG. Furthermore, we make the reasonable assumption that all SUs in an SG are identical, i.e. they have identical capacity with respect to the SIs.

5.2.2 Service Instance Protection for the 2N and No-Redundancy Models

In this section we discuss the 2N and the No-redundancy models separately and show that deciding about SI-Protection is not complex for these two cases.

5.2.2.1 The 2N Redundancy Model

In an SG with the 2N redundancy model, at most one SU will have the active HA state for all SIs and is referred to as the active SU, and at most one SU will have the standby HA state for all SIs and is usually called the standby SU. Any SU should be capable of

taking the active or the standby role for all SIs [SAF 2010d]. In order to capture unambiguously the meaning of the 2N redundancy model for an SG, we define it formally as shown in Equation 5-6. We consider any two different SUs in the SG, $su1$ and $su2$, and define two relations; the first one is for the active assignment while the second one is for the standby assignment. *ActiveAssignment* and *StandbyAssignment* are defined as relations between one SU and the set *SIList* of SIs, with the following properties:

- The *ActiveAssignment* relation is defined as a set of pairs with a range equal to the set *SIList*. Similarly, for *StandbyAssignment* relation. Therefore, each SI is taken care of once and only once, for both the active and the standby assignments.
- The capacity required, from an SU, does not exceed the SU capacity, for both the active and the standby assignments, as specified in $RequiredActiveCapacityfrom(ActiveAssignment, su1) \leq su1^{act}$ for the active part, and in $RequiredStandbyCapacityfrom(StandbyAssignment, su1) \leq su1^{stb}$ for the standby part.
- Only one SU, $su1$, is assigned the active role for all SIs and only one SU, $su2$, is assigned the standby role for all SIs, and they are different.

Equation 5-6 Formal specification of the 2N redundancy model

$$\begin{array}{l}
 \forall su1, su2 \in SList, \text{ such as } su1 \neq su2, \\
 (\exists \text{ ActiveAssignment} = \{(su1, u) \mid u \in SList\} \wedge \text{Range}(\text{ActiveAssignment}) = SList) \\
 \wedge \\
 \text{RequiredActiveCapacityfrom}(\text{ActiveAssignment}, su1) \leq su1^{act}) \\
 \wedge \\
 (\exists \text{ StandbyAssignment} = \{(su2, u) \mid u \in SList\} \wedge \text{Range}(\text{StandbyAssignment}) = SList) \\
 \wedge \\
 \text{RequiredStandbyCapacityfrom}(\text{ActiveAssignment}, su2) \leq su2^{stb})
 \end{array}$$

Having assumed that all SUs in the SG are identical, the properties specified by Equation 5-6 will be satisfied by a configuration, if and only if the SG consists of at least two SUs and anyone of these SUs is capable of taking the active or the standby role for all SIs. These necessary and sufficient conditions, summarized by Equation 5-7, can be checked easily.

Equation 5-7 Necessary and sufficient conditions for the 2N redundancy model

$$\begin{array}{l}
 |SList| \geq 2 \\
 \text{let } su \in SList \\
 (\sum_{j=1}^{|SList|} SI_j^{act}) \leq su^{act} \wedge (\sum_{j=1}^{|SList|} SI_j^{stb}) \leq su^{stb}
 \end{array}$$

5.2.2.2 The No-redundancy Model

The No-redundancy model is used for non-critical applications and components as defined in [SAF 2010d]. An SU is assigned the active HA state for at most one SI. An SI can be assigned to only one SU at a time. All SIs should be assigned if the number of SUs in service permits. An SU is never assigned the standby HA state for any SI. The No-redundancy model is formalized by Equation 5-8, where *ActiveAssignment* is simply a bijective relation between *SList* and *SList*.

Equation 5-8 Formal specification of the No-redundancy model

$$\begin{aligned}
 & (\exists \text{ActiveAssignment} = \{(su, u) \mid u \in \text{SIList}\} \wedge \text{Range}(\text{ActiveAssignment}) = \text{SIList}) \\
 & \wedge \\
 & \text{RequiredActiveCapacityfrom}(\text{ActiveAssignment}, su) \leq su^{act} \\
 & \wedge \\
 & (\forall z \in \text{SIList}, \exists!(k, z) \in \text{ActiveAssignment}) \\
 & \wedge \\
 & (\forall k \in \text{SUList}, \exists!(k, z) \in \text{ActiveAssignment})
 \end{aligned}$$

Knowing from [SAF 2010d] that any SU in the SG should be capable of protecting any SI that is protected by the SG and assuming this condition, modeled here with $\text{RequiredActiveCapacityfrom}(\text{ActiveAssignment}, su) \leq su^{act}$, is checked a priori, the only necessary and sufficient condition for an *ActiveAssignment* relation with the specified properties to exist is: $|\text{SUList}| \geq |\text{SIList}|$, and this can be checked easily. Informally, it is necessary and sufficient to have at least as many SUs in *SUList* than SIs in *SIList*.

5.2.3 Service Instance Protection for the N+M Redundancy Model

An SG with the N+M redundancy model has N+M SUs. An SU can be active for all SIs assigned to it or standby for all SIs assigned to it. In other words, no SU can be simultaneously active for some SIs and standby for some other SIs [SAF 2010d]. On the service hand, for each SI there is at most one and only one SU that is assigned the active HA state and at most one and only one SU that is assigned the standby HA state.

5.2.3.1 Formal Definition of the N+M Redundancy Model

In order to capture the characteristics of the N+M redundancy model in a precise manner, a formal specification of an SG with the N+M redundancy model is given by Equation

5-9. As for the case of the 2N redundancy model, we can distinguish two parts for expressing separately the active assignment and the standby assignments.

The 2N and the N+M redundancy models share several properties. In both cases, the SUs can only be either active or standby, and from the service side each SI should only have one active assignment and only one standby assignment. The difference is that for the N+M redundancy model, the number of SUs that are assigned the active HA state or the standby HA state is not limited to one for each. Consequently, in Equation 5-9, *ActiveAssignment* relation is a relation between as set SUs and a set of SIs; similarly for *StandbyAssignment* relation. It is well known that the 2N redundancy model is a special case of the N+M redundancy model, i.e. the 2N redundancy model can be identified as the 1+1 redundancy model.

Equation 5-9 Formal specification of the N+M redundancy model

$$\begin{aligned}
& \exists \text{ActiveAssignment} = \{(x, y) \mid x \in \text{SUList}, y \in \text{SIList} \mid \\
& \forall z \in \text{SIList}, \exists! (k, z) \in \text{ActiveAssignment} \\
& \wedge \\
& \forall w \in \text{SUList}, \text{RequiredActiveCapacityfrom}(w) \leq w^{act}\} \\
& \wedge \\
& \exists \text{StandbyAssignment} = \{(x, y) \mid x \in \text{SUList}, y \in \text{SIList} \mid \\
& \forall z \in \text{SIList}, \exists! (k, z) \in \text{StandbyAssignment} \\
& \wedge \\
& \forall w \in \text{SUList}, \text{RequiredStandbyCapacityfrom}(w) \leq w^{stb}\} \\
& \wedge \\
& \text{Domain}(\text{ActiveAssignment}) \cap \text{Domain}(\text{StandbyAssignment}) = \emptyset
\end{aligned}$$

5.2.3.2 Checking SI-Protection for an SG with the N+M Redundancy Model

In order to ensure SI-Protection at configuration time when this is not achieved by design, we need to verify the configuration against the specification given in Equation

5-9. We need a procedure to check for the properties stated in this equation. Such a procedure may have to consider all the possible combinations of SIs to assign to the SUs, and obviously it will be a complex procedure in general. In the case of the 2N redundancy model, there was only one combination of SIs, i.e. SIs are assigned all together to one SU for the active role, and all together to another SU for the standby role.

The complexity of the problem for the case of N+M can be illustrated intuitively as shown in Figure 5-2. The complexity is due to the different possible combinations of SIs we may have to consider in order to find an *ActiveAssignment* or a *StandbyAssignment* relation that satisfies the aforementioned properties. We will, in the following, show that the SI-Protection problem for the N+M redundancy model is an NP-hard problem. Therefore there is no polynomial order algorithm to solve it [Garey 1979].

According to the NP-hardness theory, a problem H is NP-hard if and only if there is an NP-complete problem L that is polynomial time Turing-reducible to an instance of H [Garey 1979]. Therefore, in order to prove the NP-hardness of SI-Protection problem, we have to find an NP-complete problem and reduce it to an instance of the SI-Protection problem.

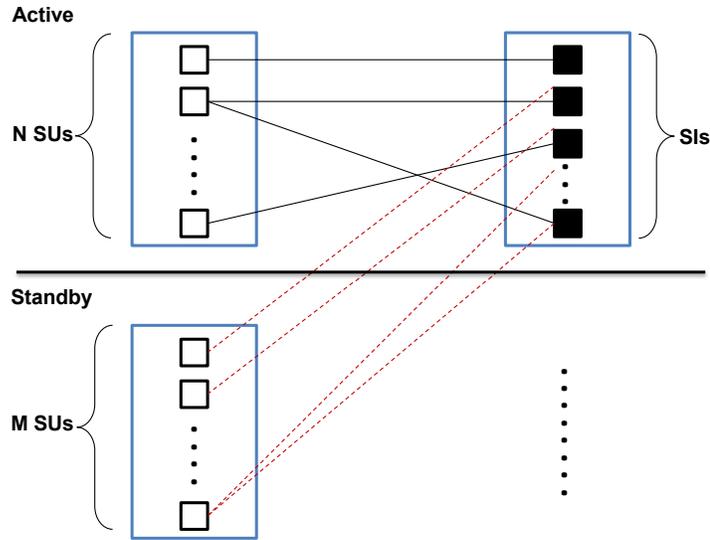


Figure 5-2 Complexity of the SI-Protection for the N+M redundancy model

For the N+M redundancy model, there is N active SUs and M standby SUs. The active and standby capacities of SUs are independent of each other, since different SUs take these different roles. Consequently, without loss of generality, we will consider here the active part only. The proof for NP-hardness for the active part can be likewise applied for the standby part. If an NP-complete problem reduces to an instance of the SI-Protection problem in polynomial time, the NP-hardness of the SI-Protection problem will be established. For this purpose, let us consider the *Subset Sum* problem, which is known to be NP-complete [Garey 1979]. The *Subset Sum* problem can be defined as follows [Garey 1979]: “Given a set of positive integers (I) and a positive integer (t), does the sum of some non-empty subset equal exactly to t ?”. To prove the NP-hardness of the SI-Protection problem, let us now consider a specific case in which the number of active SUs is 2 and each SU support only one CStype. We refer to this problem as the $(2, I)$ -assignment problem. We show the problem is NP-hard in this case; hence NP-hardness of

the general SI-Protection problem. We hereafter, present a reduction of the *Subset Some* Problem to the $(2, I)$ -assignment problem.

Theorem 1

The *Subset Sum* problem reduces to the $(2, I)$ -assignment problem in polynomial time.

Consider an instance $(I = \{a_1, \dots, a_p\}, t_1)$ of the *Subset sum* problem. Let α be the sum of members of I . Define $t_2 = \alpha - t_1$. Observe that for $t_2 < 0$, the answer to the problem is *No* and for $t_2 = 0$, the answer is *Yes*. These are trivial cases. Now, let t_2 be greater than 0 (positive). We need to define an instance of the $(2, I)$ -assignment problem. So, we have only two active SUs and the capacity of protected SIs can be represented as positive integers (they can only support one specific CStype). Let us define the capacity of SUs as $t_{max} = \max(t_1, t_2)$. Also, let the SIs have weights (a_1, \dots, a_p, β) in which $\beta = t_{max} - \min(t_1, t_2)$ (obviously they consist of CSIs of one CStype).

Lemma 1

If the answer to the *Subset sum* problem is *Yes*, then the answer to the $(2, I)$ -assignment problem is also *Yes*.

Lemma 2

If the answer to the $(2, I)$ -assignment problem is *Yes*, then the answer to *Subset sum* problem is also *Yes*.

The proof for both lemmas is straightforward. With these lemmas and based on NP-hardness theory, the NP-hardness of $(2, I)$ -assignment problem is proven. Therefore, the

NP-hardness of SI-Protection problem for the N+M redundancy model is proven. Consequently, there is no polynomial solution for this problem.

5.2.4 The N-Way-Active and N-Way Redundancy Models

An SG with the N-Way-Active redundancy model contains N SUs. Each SU has to be active for all SIs assigned to it. An SU is never assigned the standby HA state for any SI. From the service side, for each SI, one, or multiple SUs can be assigned the active HA state according to the preferred number of assignment configured for the SI. The formal specification of this redundancy model is given by Equation 5-10, where only the active assignments part is present. As for the previous case, it is defined as a relation between SUs and SIs. This relation has two properties. The first one states that each SI from the *SIList* is assigned to as many SUs as its preferred number of active assignments. The notation $z.PreferredActiveAssignments$ refers to that number. The second property is related to the capacity of the SUs, and as in the previous cases, it states that the capacity of each SU is not exceeded.

For the N-Way redundancy model, the SG also contains N SUs that protect multiple SIs. An SU can simultaneously be assigned active HA state for some SIs and standby HA state for some other SIs. At most, one SU may have the active HA state for an SI, but one, or multiple SUs may have standby HA state for the same SI. The N-Way redundancy model is formalized by Equation 5-11. The notation $z.PreferredStandbyAssignments$ refers to the preferred number of standby assignments for SI z . Notice the last property ($ActiveAssignment \cap StandbyAssignment = \emptyset$) that states that no SU is assigned active HA state and standby HA state for the same SI.

Equation 5-10 Formal specification of the N-Way-Active redundancy model

$$\begin{aligned}
 & \exists \text{ActiveAssignment} = \{(x, y) \mid x \in \text{SUList}, y \in \text{SIList} \mid \\
 & (\forall z \in \text{SIList}, \exists \text{ActiveAssignment}' \subseteq \text{ActiveAssignment}, \\
 & \text{Range}(\text{ActiveAssignment}') = \{z\} \\
 & \wedge \\
 & \mid \text{ActiveAssignment}' \mid = z.\text{PreferredActiveAssignments}) \\
 & \wedge \\
 & (\forall w \in \text{SUList}, \text{RequiredActiveCapacityfrom}(w) \leq w^{\text{act}})\}
 \end{aligned}$$

Equation 5-11 Formal specification of the N-Way redundancy model

$$\begin{aligned}
 & \exists \text{ActiveAssignment} = \{(x, y) \mid x \in \text{SUList}, y \in \text{SIList} \mid \\
 & \forall z \in \text{SIList}, \exists! (k, z) \in \text{ActiveAssignment} \\
 & \wedge \\
 & \forall w \in \text{SUList}, \text{RequiredActiveCapacityfrom}(w) \leq w^{\text{act}}\} \\
 & \wedge \\
 & \exists \text{StandbyAssignment} = \{(x, y) \mid x \in \text{SUList}, y \in \text{SIList} \mid \\
 & (\forall z \in \text{SIList}, \exists \text{StandbyAssignment}' \subseteq \text{StandbyAssignment}, \\
 & \text{Range}(\text{StandbyAssignment}') = \{z\} \\
 & \wedge \\
 & \mid \text{StandbyAssignment}' \mid = z.\text{PreferredStandbyAssignments}) \\
 & \wedge \\
 & (\forall w \in \text{SUList}, \text{RequiredStandbyCapacityfrom}(w) \leq w^{\text{stb}})\} \\
 & \text{Domain}(\text{ActiveAssignment}) \cap \text{Domain}(\text{StandbyAssignment}) = \emptyset
 \end{aligned}$$

Both the N-Way-Active and N-Way redundancy models are as complicated as the N+M redundancy model. The issue of considering different combinations of SIs remains the same. Moreover, the N-Way-Active and N-Way redundancy models allow for multiple assignment of SIs to SUs. Therefore the SI-Protection problem for both of them is at least as complex as for the N+M redundancy model. The same proof can be conducted for these two redundancy models. The SI-Protection problem for the N-Way-Active and N-Way redundancy models is also NP-hard.

5.2.5 Overcoming Complexity for Special Cases

As shown in previous sections, the SI-Protection problem is NP-hard for three redundancy models: N+M, N-Way-Active and N-Way. In order to overcome this complexity we will in this section consider a special case from the *SIList*, i.e. the set of SIs to protect, perspective. We will first explore how to reduce the complexity of the SI-Protection problem in the case of the N+M redundancy model before discussing the other two redundancy models.

Let us consider the case where *SIList* can be partitioned into subsets of identical SIs and the SIs of any pair of different subsets do not have any CStype in common. We refer to this as the case of CStype_Disjoint subsets of identical SIs. More precisely, *SIList* can be partitioned into *SISubSet₁*, *SISubSet₂*, ..., and *SISubSet_n*, where each *SISubSet_i* contains only identical SIs and *SISubSet_i* and *SISubSet_j* do not have any CStype in common when $i \neq j$.

For the N+M redundancy model, any SU in the SG can either be assigned the active or standby HA state. From the service perspective, for each SI, we only have one active assignment and one standby assignment. Consequently, we can divide the set of SUs into two partitions: the active and standby partitions. Any SU in the active partition acts only as active and any SU in the standby partition acts only as standby.

We assumed that SUs in an SG are all identical, which means they all have the same number of components of the same component types. We have so far defined and discussed the capacity in terms of CStypes, we will here define another capacity for an SU with respect to SIs as the number of SIs that the SU can provide service for at the

same time. In fact, each SU can have an active capacity and a standby capacity with respect to each SI. We determine the active and standby capacity of an SU with respect to each SI using the division operation introduced in Section 5.2.1 as given by Equation 5-12.

Equation 5-12 Active/Standby capacity of an SU w.r.t. to an SI

$Integer\ c : ActiveCapacity(su : SU, si : SI)$ $Let\ DivisionSet : Set(Integer) = su^{act} \div si^{act}$ $c = Min(DivisionCap);$ $Integer\ c : StandbyCapacity(su : SU, si : SI)$ $Let\ DivisionSet : Set(Integer) = su^{stb} \div si^{stb}$ $c = Min(DivisionSet)$

The set of protected SIs, $SIList$, is partitioned into $CSType_Disjoint$ subsets of identical SIs. By calculating the capacity of one SU for one of the SIs of each partition we will have capacities of any SU in the SG regarding any SI in the $SIList$. We know that $SIList = SISubSet_1 \cup SISubSet_2 \cup \dots \cup SISubSet_n$, and each $SISubSet_i$ is $CSType_Disjoint$ with the other subsets. Consequently, we can define an ordered set of n integers for an SU in the SG: $\{AC_1, AC_2, \dots, AC_n\}$, in which AC_i represents the active capacity of the SU with respect to the SIs in $SubSet_i$. Similarly, we define a set of integers for each SU in the SG as $\{SC_1, SC_2, \dots, SC_n\}$, in which sc_i represents the standby capacity of the SU with respect to the SIs in $SISubSet_i$. Now, we have all required information in order to check whether an SG with the N+M redundancy model is capable of protecting the set of SIs it is configured for, or not.

As mentioned earlier, in the N+M redundancy model, we have N SUs and M SUs that are taking the active assignments and standby assignments, respectively. From the service perspective, *SIList*, the list of protected SIs, is partitioned into n CStype_Disjoint subsets of identical SIs. In this specific situation, the conditions specified in Equation 5-13 represent the necessary and sufficient conditions for the SG to protect the set of SIs it is configured for.

Equation 5-13 Necessary and sufficient conditions for the N+M redundancy model

$$\boxed{\begin{array}{l} \forall 1 \leq i \leq n, AC_i \times N \geq |SISubSet_i| \\ \wedge \\ \forall 1 \leq i \leq n, SC_i \times M \geq |SISubSet_i| \end{array}}$$

Intuitively, $\forall 1 \leq i \leq n, AC_i \times N \geq |SISubSet_i|$, states that there is enough capacity in the SUs of the SG to protect all the SIs in *SISubSet_i*, each SI once. Since the *SISubSets* are CStype_Disjoint with each other, each SU will be able to provide service for all the subsets simultaneously. The same reasoning applies for the standby part. Moreover, the last property of the N+M redundancy model is satisfied, since we handle active and standby SUs separately. A simple procedure can be written for checking the conditions in Equation 5-13.

One very specific case for AMF configurations is when all SIs in the *SIList* are identical. This is actually a special case of the CStype_Disjoint subsets of identical SIs, with the number of subsets equal to one. Another very specific case is when all SIs in *SIList* are CStype_Disjoint with each other. In other words, they are composed of CSIs that do not have any CStype in common. This is another special case of the CStype_Disjoint subsets of identical SIs, where *SIList* is partitioned into n subsets of cardinality one.

Similar conditions and reasoning can be followed for the N-Way-Active and N-Way redundancy models. In the case of N-Way-Active redundancy model, let us assume the number of SUs in the SG is N . We consider again the first condition in Equation 5-13, but now taking also into account the number of preferred active assignment for each SI. Indeed, the preferred number of active assignments for each SI has to be taken into account as factor for the required capacity and we can check that SUs in the SG have the required capacity to protect the SIs. However, the problem in this case is how to make sure that an SI is not taken care of twice by the same SU? Therefore, we add the following condition:

$\forall 1 \leq i \leq n, N \geq MaxPrefAct$ in which $MaxPrefAct$ is the highest number among the preferred numbers of active assignments for the SIs in $SISubSet_i$.

This condition is necessary and sufficient to ensure that a given SI can be assigned to as many different SUs as specified by its preferred number of active assignments, knowing that all SUs in the SG are identical. These necessary and sufficient conditions are simple to check.

In the case of the N-Way redundancy model, let us also assume N as the number of SUs in the SG. The first condition of Equation 5-13 remains the same as only one active assignment is required per SI. The second condition is modified, M replaced by N , and to take into account the preferred number of standby assignments for the SIs and make sure the SUs have the capacity to protect the SIs in the standby role. Similarly to the N-Way-Active redundancy model, we need another condition to make sure that an SU is not assigned more than once the standby HA state for a given SI. Moreover, an SU should

not be assigned the active HA state and the standby HA state for a given SI. We therefore add the following condition: $\forall 1 \leq i \leq n, N \geq MaxPrefStb + 1$ in which $MaxPrefStb$ is the highest number among the preferred numbers of standby assignments for the SIs in $SISubSet_i$ to ensure there is enough SUs for standby and active assignments, knowing that all SUs in the SG are identical.

5.2.6 Overcoming Complexity with Heuristics: Checking for Service Protection Using Heuristics

In the previous section, we proved that in the case of N+M, N-Way and N-Way-Active redundancy models the problem is NP-hard in general. For these three redundancy models, we identified some specific situations where the problem can be simplified. In this section, we tackle the problem further and propose a solution for the N+M, N-Way and N-Away-Active redundancy models that is based on heuristics. Our solution is based on extensions to the well-known problem of bin-packing [Coffman 1996]. We replace bins and objects with SUs and SIs, respectively. We consider different types of capacity, i.e. capacity vector, unlike the single type of capacity in the classical bin-packing problem.

The bin-packing problem has already been revisited and extended to vector bin-packing, see for instance [Csirik 1990, Patt-Shamir 2010, Rao 2010]. Vector bin-packing is a variation of classical bin-packing in which the capacity of bins and objects is described in terms of a vector of capacities [Csirik 1990]. Several approximation algorithms have been proposed to optimize the number of bins. Recently, Patt-Shamir and Rawitz explored the vector bin-packing problem with bins of variable sizes and presented an approximation algorithm [Patt-Shamir 2010]. In [Rao 2010] Rao et al. developed an

approximation algorithm based on the near-optimal solution of linear programming relaxation of integer programming. These approximation algorithms introduce a boundary guaranteeing that their sub-optimal result will not exceed this boundary. This boundary is expressed as a factor of the optimal solution and the parameters (number of objects and the size of the vector) of the problem. Furthermore, the amount of computational and memory resources necessary for solving the problem will increase exponentially when the boundary becomes close to the optimal solution. For this reason, the efficiency of these approximation algorithms will rarely prove to be practical for large systems such as AMF configurations. Heuristics, however, target reasonably good solutions efficiently [Pearl 1984]. Moreover, the main concern in the abovementioned papers is the approximation of the optimal number of bins, while in our case we are interested in finding a possible assignment of a given set of SIs to a given set of SUs. Therefore, based on the traditional bin-packing problem heuristics, we devised new heuristics for solving the SI-Protection problem taking into account the specificities of the domain in question, i.e. SUs, SIs, and redundancy models.

We extend the three well-known heuristics for bin-packing. Each of these extensions takes the *SUList* and *SIList* as input and decides if there exists a way to assign all SIs of the *SIList* to the SUs of the *SUList*. If an algorithm succeeds in assigning all SIs to the SUs, the answer to the problem is ‘Yes’. If it fails, the answer could be ‘Yes’ or ‘No’. Since all these algorithms take a sequence of SIs and assign them one by one, an algorithm will answer ‘No’ if it fails to assign an SI at a certain point. This may be a False negative. When all SIs are successfully assigned to the SUs, the algorithm returns ‘Yes’ as result. Therefore, the signature of each algorithm can be represented as:

Boolean bin_packing_extension_x(SUList, SList)

It is worth noting that these extensions are generic algorithms for deciding about the SI-Protection and do not consider any specific redundancy model. In Section 5.2.6.4, we discuss the application of these algorithms to each of the redundancy models.

To achieve better results, our approach applies all proposed algorithms to the sequence and then determines the logical OR of the answers. Since these algorithms are different (and somehow based on opposite principles), the probability of a False negative result is reduced.

5.2.6.1 First-Fit approach (FF)

The first approach is the First-Fit (FF) approach, where we preserve a fixed order of SUs in the *SUList* during the whole processing. To assign a given SI to an SU, we simply take the first available SU in the *SUList* which can serve the SI.

Although the FF approach appears to be the easiest heuristic to the problem, it is known to be quite effective for $k = 1$ (classical bin-packing).

Complexity: The assignment of each SI to each SU can be achieved with k comparisons between the provided and required capacities of the SU and the SI. Moreover, the number of SUs that need to be checked before finding the appropriate one can reach n , at the most. Considering the number of assignments which equals the number of SIs (m), the complexity of this approach is $m \times n \times k$ in the worst case.

5.2.6.2 Best-Fit approach (BF)

This approach gives the best results in practice for the classical bin-packing problem [Kenyon 1996]. We keep the SUs sorted in an increasing order of remaining capacities, and find the first SU capable of handling the load of the SI. Therefore, a given SI is assigned to an SU which has the minimum remaining capacity among those which have enough capacity for the SI under consideration. Note that the list of SUs should be sorted after each assignment. Here, the goal is to exhaust an SU as much as possible before moving to the next. BF is occasionally referred to as *unbalanced assignment approach* [Kenyon 1996]. Since there is no single value defined as the ‘capacity’ of each SU, the provided capacity being represented through a list of non-negative integers, it is necessary to come up with a single criterion for the capacity of each SU, and to sort the SUs in the *SUList* based on this criterion. In what follows, we introduce three different criteria to represent the capacity of a given SU.

Total Capacity

Given the remaining capacity list ($\langle c_1^i, \dots, c_k^i \rangle$) for a given SU (SU_i), the total capacity is the sum of the remaining capacities of all supported CTypes in SU_i (*capacity of $SU_i = \sum_{t=1}^k c_t^i$*).

For instance, let us consider the example of Figure 3 where we have three SUs (SU_1, SU_2, SU_3) supporting three different CTypes and let the remaining capacity list for these SUs be $\langle 4, 2, 1 \rangle$, $\langle 1, 1, 1 \rangle$, and $\langle 2, 4, 0 \rangle$, respectively. The total capacities for the SUs are 7, 3, and 6, resulting in the sorted list $\{SU_2, SU_3, SU_1\}$. On the service side, there are two unassigned SIs (SI_1, SI_2) with the required capacity list of $\langle 1, 2, 0 \rangle$ and $\langle 3, 2, 1 \rangle$,

respectively. For assigning SI_1 , SU_2 will be considered first, then SU_3 and finally SU_1 . SU_1 does not have the required capacity of each CStype, however SU_3 does in fact have this capacity.

Complexity: Sorting the $SUList$ can be achieved in $O(n \log n)$ and keeping it sorted is $O(\log n)$. For each SI, we need to examine at the most all the n SUs in the $SUList$ in order to find the proper SU. This can be done in $n \times k$ comparisons. In addition, after the successful assignment of an SI, we need to keep the $SUList$ sorted. As a result, the complexity of this approach is $m \times (n \times k + O(\log n)) + O(n \log n)$.

As a variation for this case, one may also consider the sorting of the SIs at the beginning of the process, according to the total required capacity and processing the SI with the smallest capacity first or last. However, sorting SUs or SIs according to total provided or required capacity, respectively, does not necessarily help as it does not look into CStype capacities which are important for the assignments.

Relative Capacity

Contrary to the total capacity criterion, the relative capacity is defined with respect to a specific SI and is based on the largest element of the required capacity list of the SI. As a result, for each SI, the sorted list of SUs may differ. For a given SI (SI_j) with the capacity list of $\langle r_1^j, \dots, r_k^j \rangle$, let the index of the largest member of the required capacity list be $t = \text{argmax}(\langle r_1^j, \dots, r_k^j \rangle)$. This means that, for SI_j , the number of CSIs of CStype $_t$ is larger than the number of CSIs of the other CStypes. Consequently, for SI_j we need to sort the $SUList$ based on the c_t of each SU (e.g. c_t^i for SU_i).

Let us consider again the example in Figure 3. The largest required capacities of SI_1 and SI_2 are 2 and 3, respectively. Therefore, the relative capacity criterion for SI_1 is c_2 , which results in the sorted $SUList$, $\{SU_2, SU_1, SU_3\}$. Similarly, c_1 is the criterion for SI_2 and the sorted $SUList$ is $\{SU_2, SU_3, SU_1\}$.

Complexity: The complexity of the approach is very similar to the case of total capacity. The only difference is that the sorted list of SUs is different for each SI and thus, we need to sort the $SUList$ for each SI separately. Consequently, the complexity of this approach is $m \times (n \times k + O(n \log n))$.

Critical Capacity

Similar to the relative capacity, this criterion is also defined with respect to each SI. Here our objective is to find the most critical CStype for each SI and then sort the list of SUs based on this criterion. The most critical CStype for each SI is the CStype which has the largest required capacity in the SI while having the smallest provided capacity among the SUs in the $SUList$. To this end, we first determine the total capacity per CStype of the SUs as $\langle tc_1, \dots, tc_k \rangle = \sum_{i=1}^n \langle c_1^i, \dots, c_k^i \rangle = \langle \sum_{i=1}^n c_1^i, \dots, \sum_{i=1}^n c_k^i \rangle$.

Thereafter, for a given SI_j with the required capacity list of $\langle r_1^j, \dots, r_k^j \rangle$, the index of the most critical required capacity is:

$$T = \arg \max \left(r_1^j / tc_1, \dots, r_k^j / tc_k \right)$$

Consequently, for SI_j we need to sort the $SUList$ based on the c_T of each SU (e.g. c_T^i for SU_i).

Going back to the example in Figure 3, the total capacity per CStype of the *SUList* is $\langle 7, 7, 2 \rangle$. For SI_1 based on the calculation $(\langle \frac{1}{7}, \frac{2}{7}, \frac{0}{2} \rangle)$, the critical required service is r_2 and hence, the *SUList* should be sorted according to c_2 , which results in the sorted list $\{SU_2, SU_1, SU_3\}$. With the same calculation, the *SUList* for SI_2 is sorted based on c_3 and results in $\{SU_3, SU_2, SU_1\}$.

Complexity: The complexity of the critical capacity is the same as for relative capacity, i.e. $m \times (n \times k + O(n \log n))$.

5.2.6.3 Worst-Fit approach (WF)

While this algorithm is not preferred in practice to the BF approach, it is important as it uses a contrary approach, and occasionally gives positive answers when BF fails. The algorithm is more or less the same as for the BF approach the only difference being that the *SUList* is sorted in a decreasing order of capacities. In fact, the algorithm attempts to assign SIs to the SUs in a balanced way. To sort the *SUList*, we can use the exact same sorting criteria as described for the BF approach in 5.2.6.2.

5.2.6.4 Taking Into Account the Redundancy Models

In the previous section, we introduced three different approaches for checking the protection of the SIs. In addition, we have also defined three different criteria for sorting the list of SUs that can be used for both the BF and the WF approaches. Therefore, we presented seven different heuristic methods for solving the SI-Protection problem that can be applied in sequence to improve the accuracy of the solution. However, all presented approaches target the generic case of the SI-Protection without taking into account the features and the specific constraints of the redundancy models. In this

section, we discuss how we map these general approaches for the different redundancy models, N+M, N-Way, and N-Way-Active.

The N+M Redundancy Model

In the N+M redundancy model, N SUs support the active assignments and M SUs support the standbys. This model allows at the most one active and one standby assignment for each SI. Assuming that the standby SUs are distinguished from active SUs, we apply our approach, the sequence of seven heuristic methods defined previously, for the N SUs configured to support the active assignment, considering their active capacity. Thereafter, we apply the approach for M SUs configured to support the standby assignment, considering their standby capacity. We are certain that the SG can protect the SIs if and only if the result of the method is ‘Yes’ for both N active SUs and M standby SUs. Please note that if a “No” answer results for either case, this may be a False negative.

The N-Way-Active Redundancy Model

An SG with the N-Way-Active redundancy model has N SUs which are assigned only as active and has no SU assigned as standby. Furthermore, each of the SIs protected by this SG can be assigned to more than one SU as specified in the *PreferredActiveAssignments* configuration attribute. In previous sections we discussed one assignment per SI only. In order to handle multiple assignments, whenever we consider an SI, we assign it to *PreferredActiveAssignments* different SUs before proceeding to the next SI. Every assignment is handled according to the methods in Section 5.2.6.1, Section 5.2.6.2, and Section 5.2.6.3.

N-Way Redundancy Model

An SG with the 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 SU while it can be assigned standby to several SUs (as specified in the *PreferredStandbyAssignments* attribute). The solution for this redundancy model is quite similar to the one for N-Way-Active. For the single active assignment in N-Way redundancy model, we consider the active capacity of the SUs while, for multiple standby assignments, the standby capacity of the SUs is taken into account. The same SU cannot be reassigned to the same SI, neither as standby nor as active.

5.2.6.5 Incremental Design of AMF Configurations

The previously specified validation technique assigns the SIs to the SUs and returns ‘No’ if it fails to do so for any SI. In this case we propose to modify the invalid SG by adding resources, namely SUs incrementally, to increase the provided capacities.

At the point where the technique fails to assign an SI, we add SUs to the *SUList* and continue the assignment process. This process continues until all SIs are assigned or until it again fails to assign a certain SI and requires additional SUs. At the end of this incremental process, all SIs must be assigned to the SUs in the augmented *SUList*. The number of additional SUs to be added each time the algorithm fails in assigning a given SI depends on the redundancy model of the SG and in some cases on other configuration attributes.

In the case of the active part of the N+M and N-Way redundancy models only one SU should be added. For the N-Way-Active redundancy model and the standby part of the N-

Way, the number is equal to the number of remaining active/standby assignments of the SI in question i.e., if Q assignments of an SI have already taken place before the failing point, the number of additional SUs is equal to $PreferredActiveAssignments - Q$ or $PreferredStandbyAssignments - Q$. More specifically, one SU for handling the standby assignment will be added in the case of N+M and $PreferredStandbyAssignments$ SUs will be added in the case of the N-Way redundancy model.

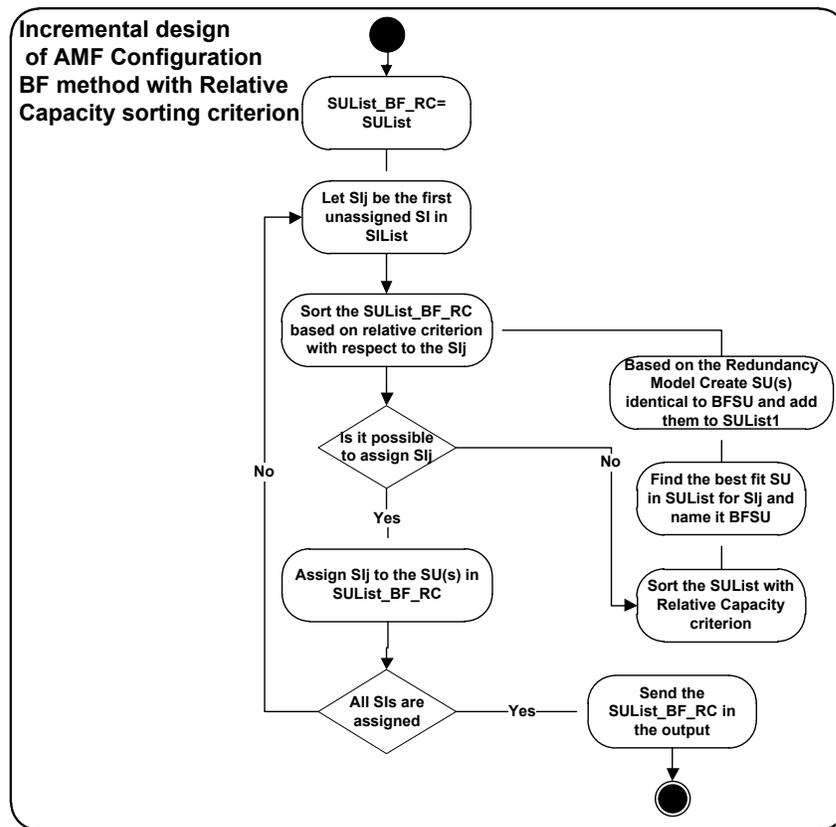


Figure 5-3 Incremental AMF configuration design using BF method with relative capacity sorting criterion

The creation of the additional SU(s) varies depending on the applied heuristic method used. More specifically, in the BF method the extra SU(s) for a given SI is/are identical to the first SU in the sorted (increasing order) list of SUs in the *SUList*. However, for a given SI in the WF method, the additional SU(s) is/are identical to the first SU in the

sorted (decreasing order) list of SUs in the *SUList*. In other words, the additional SU(s) for a given SI is/are identical to the best fit SU in the BF method and identical to the worst fit SU in the WF method. In order to sort the *SUList*, we use the same sorting criteria as used in the heuristic methods.

It is worth noting that, for the case of the FF approach, the extra SU is simply identical to the first SU in the *SUList* (i.e. the first fit SU). Figure 5-3 shows the activity diagram for the AMF configuration incremental design method using BF method with the relative capacity as sorting criterion.

In order to illustrate our incremental design approach, let us add three more SIs, $SI_3 = \langle 3,2,1 \rangle$, $SI_4 = \langle 2,1,0 \rangle$, and $SI_5 = \langle 0,1,0 \rangle$ to the example in Figure 3. The *SIList* becomes $\{SI_1, SI_2, SI_3, SI_4, SI_5\}$ with the required capacity list $\{\langle 1,2,0 \rangle, \langle 3,2,1 \rangle, \langle 3,2,1 \rangle, \langle 2,1,0 \rangle, \langle 0,1,0 \rangle\}$, while the *SUList* remains the same. In this example, we use the BF method and we apply the relative capacity criterion for sorting the *SUList*. Figure 5-4 shows the steps of the approach. As shown in part (2) of Figure 5-4, the *SUList* is sorted according to the relative capacity criterion of SI_1 (i.e. c_2) in an ascending order. Afterwards, the algorithm finds the first SU in the sorted *SUList* which has the adequate capacity to support SI_1 , SU_1 , in this case. After the successful assignment of SI_1 , the algorithm proceeds to SI_2 by sorting the *SUList* according to the relative capacity of SI_2 and by finding the appropriate SU to support it (part (3) of Figure 5-4). As presented in part (4) of Figure 5-4, after sorting the *SUList*, the algorithm succeeds in assigning SI_3 to SU_3 . For SI_4 , after sorting the *SUList*, the algorithm fails to find an appropriate SU capable of supporting SI_4 . This means that the SG cannot protect the set

of SIs configured for it and thus the configuration is “likely” not valid. In this case, the algorithm proceeds by adding an extra SU in order to increase the capacity. To do so, the algorithm determines the best fit SU among the SUs of the original *SUList* (see part (1) of Figure 5-4) and creates an SU with the same capacity, adding it to the *SUList*. As presented in part (6) of Figure 5-4, SU_4 is created based on the SU_2 and is added to the *SUList* in order to support the load of SI_3 . The remaining capacity of the *SUList* is sufficient to support the load of SI_5 and therefore it is assigned to SU_4 see part (8) of Figure 5-4).

In the last row of Figure 5-4, part (9) represents the remaining capacity of the *SUList* after the successful assignment of the entire *SIList* and part (10) shows the order of the active assignment of each SI to one of the SUs of the augmented *SUList*.

In order to get the best result, we run seven different heuristics in parallel. Each one will end up with an *SUList*, and the final *SUList* will be the list with the least number of SUs. In other words, the final result will be the *SUList* with minimal additional SUs and therefore, the resources used for protecting services will be relatively minimized. In the case of equality between at least two lists, one may choose the list of SUs with minimal total capacity or the list with maximal total capacity, depending on the design criteria of minimizing resources further or on extendibility. However, comparing lists of SUs with different capacities is not straightforward and further investigations are required. Notice that having a smaller number of SUs will facilitate the management of the availability of the applications by the AMF middleware, resulting in the increase of protection level given a fixed number of deployment nodes. Obtaining the original *SUList* as the final result indicates that the input SG is valid and can protect its SIs without any additional

SUs. Figure 5-5 presents the overview of our approach for the incremental design of AMF configurations.

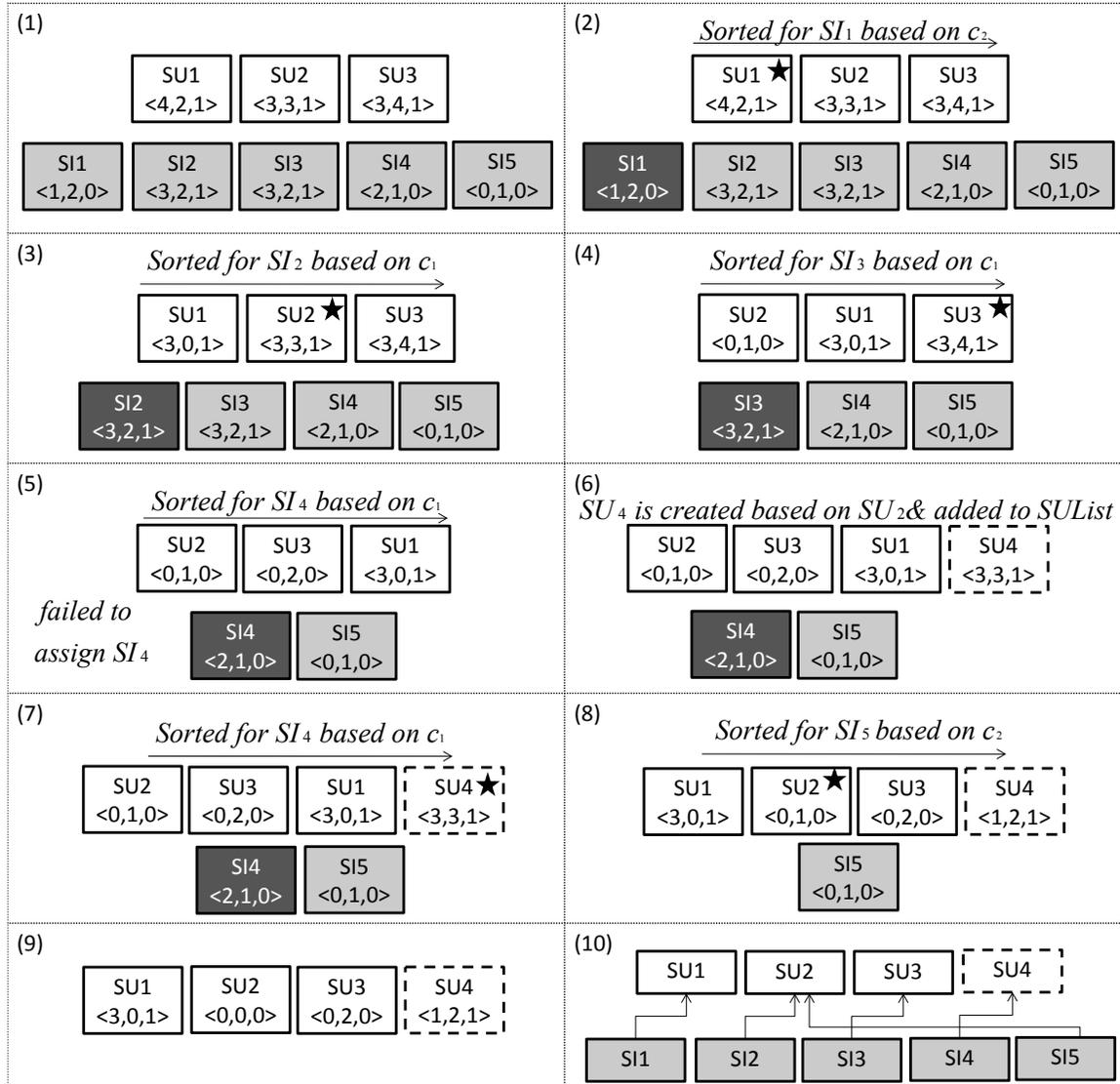


Figure 5-4 An example for the incremental design approach

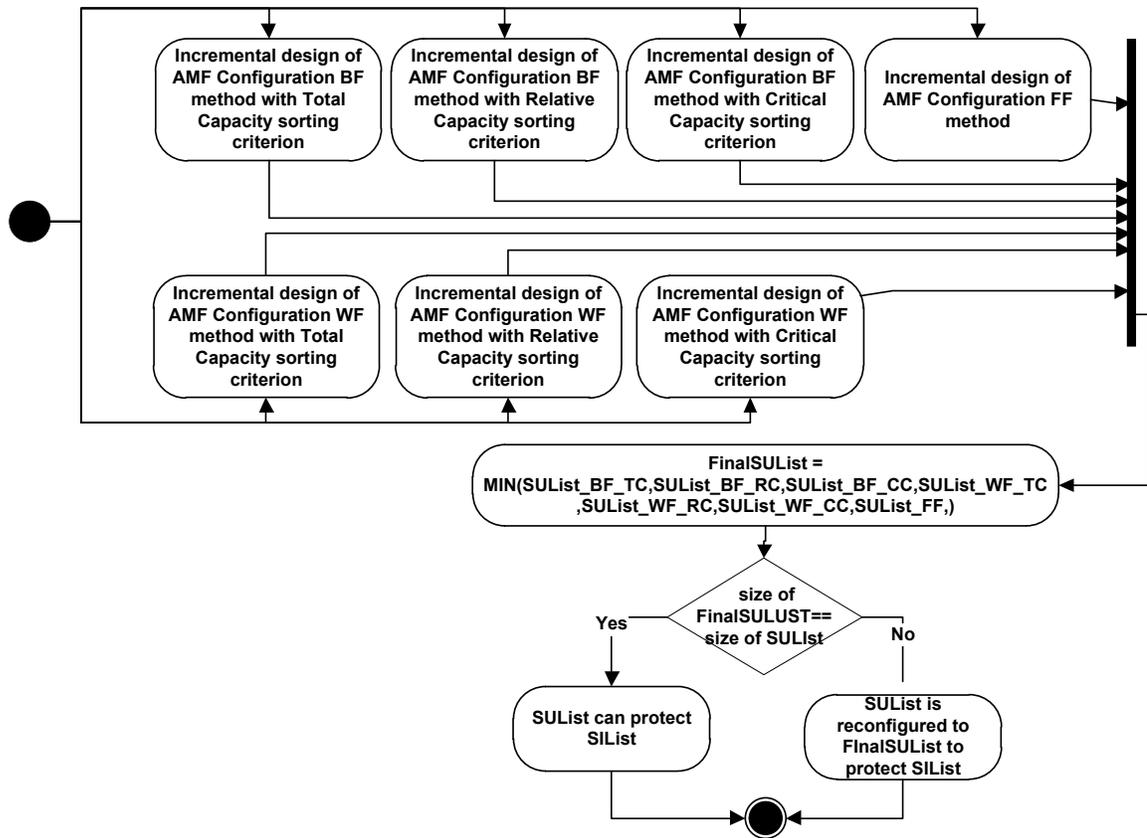


Figure 5-5 Overview of the incremental design approach

5.3 Summary

In this chapter we presented our approach for a validation of third-party AMF configurations which handles both the syntactical and the semantic validations of these configurations. For syntactical validation, our approach includes the mapping of a given third-party configuration—represented in the IMM (Information Model Management) XML format—to an instance of the AMF sub-profile. During this mapping, the consistency of the configuration with respect to the standard specification of the AMF middleware is checked.

In semantic validation, we focused on the alignment of AMF configurations with the protection level expressed through the characteristics of configuration elements. Ensuring

the protection of the services at configuration time, as required and according to the specified redundancy model, is proved to be NP-hard for most redundancy models. To tackle this problem, we have presented a heuristics based approach by extending the heuristics introduced for the well-known bin-packing problem. The precision of the approach is enhanced by embedding seven different heuristic methods in order to obtain better results. In terms of performance, we have tested our approach on a limited number of small scale configurations. However, analysing the performance and the accuracy of the approach is a complex task which requires the implementation of a simulation framework for different scenarios, a task which is left for future work in this research stream. As a corollary, we proposed a technique for the incremental modification of “likely” invalid configurations into valid ones. We believe that our technique may lead to over-dimensioned systems, though only by adding a minimal number of extra resources.

Chapter 6

Model-based AMF Configuration Generation

As mentioned in the Chapter 1, the model-driven paradigm helps in managing the complexity of the generation process by raising the level of abstraction at which the configuration properties have to be defined. This allows for both the simplification of the generation process and for the reduction of potential errors and/or inconsistencies. Moreover, handling configuration generation in a high level of abstraction improves the maintainability of the approach compared to the code-centric approaches presented in [Kanso 2008, Kanso 2009].

The content of this chapter has been published in [Salehi 2010b and Salehi 2011b].

6.1 Overall View

The model-driven AMF configuration generation approach consists of a set of transformation rules among models that are instances of the previously described profiles. Starting from the description of software expressed through an ETF model, this approach generates an AMF configuration which is an instance of the AMF profile. Moreover, the approach considers the requirements of the configuration specified by configuration designer. Configuration requirements specify the set of services to be provided by a given software system through the target AMF configuration. More specifically, they define the

different characteristics of the services, such as their types, the number of instances of a certain service type, the relationships between services, and the level of protection expressed in the context of AMF in the form of redundancy models.

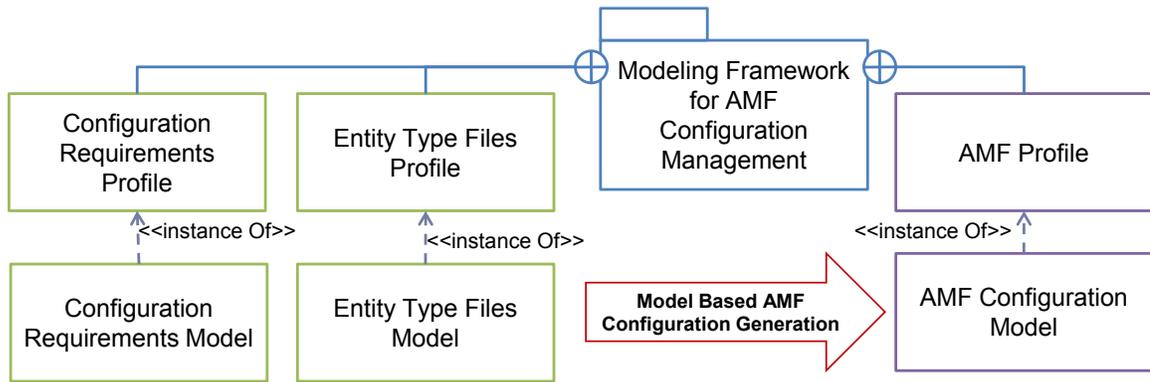


Figure 6-1 The overall process of model-based AMF configuration generation

Figure 6-1 illustrates the different artefacts involved in the generation process. The input for the transformation consists of configuration requirements and the description of software to be protected, while the output of the transformation is an AMF configuration for the software that satisfies the configuration requirements. The inputs and outputs are modeled as instances of different profiles.

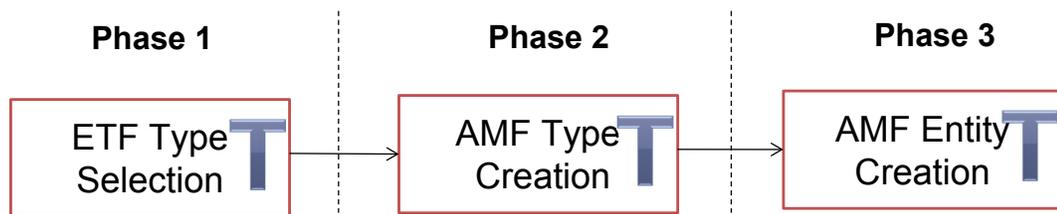


Figure 6-2 The main phases of the model transformation approach

This process consists of a set of transformation rules expressed in a declarative style defined among different elements of our profile. AMF configurations are generated by applying the transformation rules to the model elements representing software entities

and configuration requirements. These rules, implemented using ATL, abstract from the operational steps that have to be performed in order to generate the target elements. However, the rules presented in this chapter only focus on a high level view of the stereotypes, tagged definitions, and relationships between the elements, hiding the implementation details in order to improve readability.

As shown in Figure 6-2, the transformation process has three distinct phases, namely, 1) the selection of the software to be used to satisfy the requirements, 2) the creation of proper AMF entity types based on the selected ETF types, and 3) the instantiations of AMF entities related to each AMF entity types. More precisely, the configuration generation method proceeds with selecting the appropriate ETF types for each service specified by the requirements. Therefore, the selected software is used to derive the AMF types and to instantiate the AMF entities that will compose the configuration. For each transformation phase, Figure 6-3 illustrates the input and output models and their referenced metamodels.

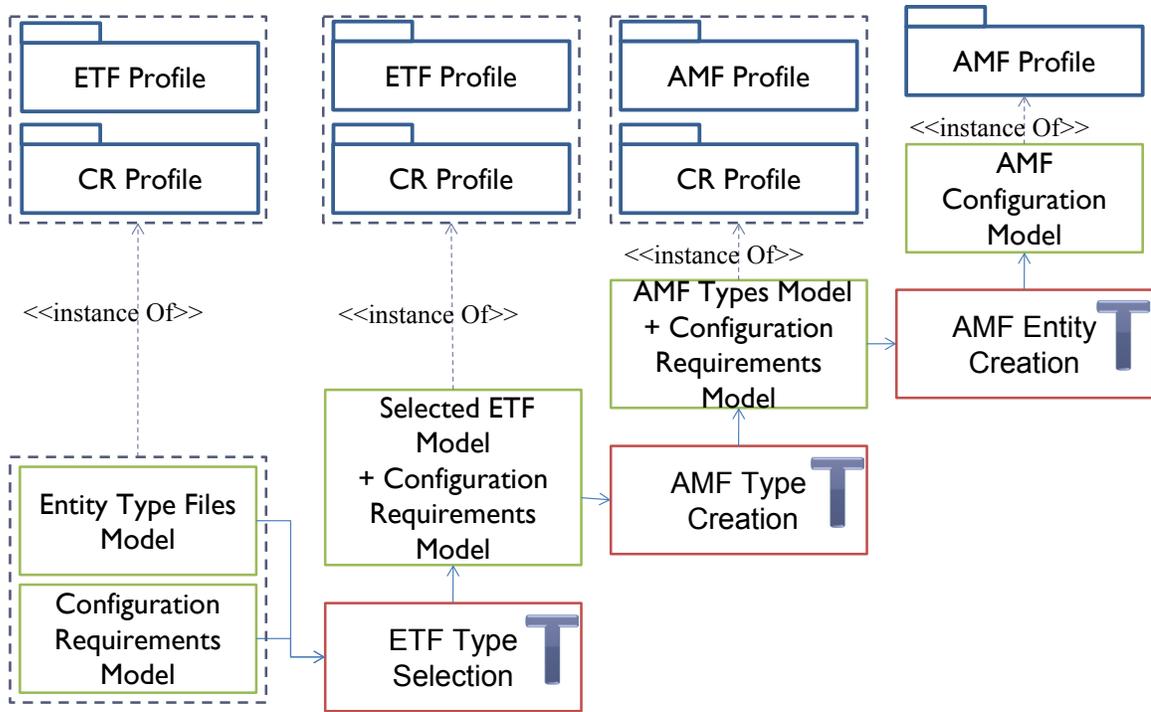


Figure 6-3 The relation between the models and the transformation phases

During the model-driven generation of AMF configurations a set of relationships and attributes are temporarily necessary to link the elements of different sub-profiles. The relationships are used to navigate the models involved in the transformation activities in order to retrieve all the information that is required to generate an AMF configuration. These relationships are modeled in terms of UML associations between the elements of the CR sub-profile on one side and elements of the ETF and AMF sub-profiles on the other side. Table 6-1 presents the list of associations and their descriptions. The variables used to store temporary information used in several steps of the generation approach are modeled in terms of attributes of the CR's model elements. Table 6-2 specifies the list of these attributes.

Table 6-1 The list of the associations that model the relationships among elements of the sub-profiles

Source Element	Target Element	Role Name	Multiplicity	Description
MagicCrCsiTemplate	MagicEtfCompType	properEtfCt	[0..n]	Refers to the ETF Component Types which are selected for this CSITemplate in the process of configuration generation
MagicCrCsiTemplate	MagicSaAmfCompType	properAmfCt	[0..n]	Refers to the AMF component types which are created for this CSITemplate in the process of configuration generation
MagicCrCsiTemplate	MagicSaAmfComp	properAmfComp	[0..n]	Refers to the AMF components which are created for this CSITemplate in the process of configuration generation
MagicCrSiTemplate	MagicEtfSUType	properEtfSUT	[0..n]	Refers to the ETF SUTypes which are selected for this SITemplate in the process of configuration generation
MagicCrSiTemplate	MagicSaAmfSUType	properAmfSUT	[0..n]	Refers to the AMF SU types which are created for this SITemplate in the process of configuration generation
MagicCrSiTemplate	MagicSaAmfSU	properAmfSU	[0..n]	Refers to the AMF SUs which are created for this SITemplate in the process of configuration generation
MagicCrSgTemplate	MagicEtfSGType	properEtfSGT	[0..n]	Refers to the ETF SGTypes which are selected for this SGTTemplate in the process of configuration generation

It is important to mention that the CR model requires processing before starting any of the abovementioned transformation phases. This pre-processing activity consists of setting the initial values of the attributes specified in Table 6-2. These attributes will be used throughout this chapter in several transformation steps. The goal of this activity consists of determining the expected load of the SIs of each SI template that an SU of the SG protecting those SIs will handle. This is motivated by the fact that ETF types may

specify capacity limitations of Component Types and SUTypes articulated into three steps:

1. Calculation of the number of SGs that are allowed to protect the SIs of a particular SG template.
2. Calculation of the number of SIs from each SITemplate that will be assigned to each SG. The calculation is based on the number of SGs calculated in Step 1. This step initializes the value of the attribute *expectedSIsperSG*.
3. Calculation of the load of SIs that each SU of the SG is supposed to support initializing the value of the attributes *activeLoadperSU* and *stdbLoadperSU*. The calculation is based on the minimum number of SIs an SG must handle calculated in Step 2.

The entire process is implemented as a refinement ATL rule on the SITemplate element of the CR model.

```
rule CR_Preprocessing {
  from
    s: MagicCRProfile!MagicCrRegularSiTemplate

  using{

    --Calculates the number of SGs

    maxNumSGs : Integer =
    s.magicCrBelongsToSgTemplate.magicCrGroupsSiTemplates
    ->iterate(sit, min:Integer = 0|
    if sit.magicCrRegSiTempNumberofSis/sit.magicCrRegSiTempMinSis > min
    then
    min= sit.magicCrRegSiTempNumberofSis/sit.magicCrRegSiTempMinSis
    endif);

    --Calculates the number of expected SIs per SG
    SIperSG : Integer =
    s.magicCRRegSiTempNumberofSis/maxNumSGs +1

  }

  to
```

```

t: MagicCRProfile!MagicCrRegularSiTemplate(

  expectedSIperSG <- SIperSG,

  --Calculates the active load per SU based on the required redundancy
  model
  activeLoadperSU <-
  if (s. magicCrBelongsToSgTemplate. magicCrSgTempRedundancyModel =
    'SA_AMF_N_WAY_REDUNDANCY_MODEL'
  or
    s. magicCrBelongsToSgTemplate. magicCrSgTempRedundancyModel =
    'SA_AMF_N_WAY_ACTIVE_REDUNDANCY_MODEL')
  then
    ceil((SIperSG* s. magicCrSiTempNumberofActiveAssignments)/
    (s. magicCrBelongsToSgTemplate. magicCrSgTempNumberofActiveSus-1))
  elseif
    (s. magicCrBelongsToSgTemplate. magicCrSgTempRedundancyModel =
    'SA_AMF_2N_REDUNDANCY_MODEL'
  or
    s. magicCrBelongsToSgTemplate. magicCrSgTempRedundancyModel =
    'SA_AMF_NPM_REDUNDANCY_MODEL')
  then
    ceil(SIperSG/ s. magicCrBelongsToSgTemplate.
    magicCrSgTempNumberofActiveSus)
  elseif
    (s. magicCrBelongsToSgTemplate. magicCrSgTempRedundancyModel =
    'SA_AMF_NO_REDUNDANCY_MODEL')
  then
    1
  endif ,

  --Calculates the standby load per SU based on the required redundancy
  model

  stdbLoadperSU <-
  if (s. magicCrBelongsToSgTemplate. magicCrSgTempRedundancyModel =
    'SA_AMF_N_WAY_REDUNDANCY_MODEL'
  or
    s. magicCrBelongsToSgTemplate. magicCrSgTempRedundancyModel =
    'SA_AMF_N_WAY_ACTIVE_REDUNDANCY_MODEL')
  then
    ceil((SIperSG* s. magicCrSiTempNumberofStdbAssignments)/
    (s. magicCrBelongsToSgTemplate. magicCrSgTempNumberofActiveSus-1))
  elseif
    (s. magicCrBelongsToSgTemplate. magicCrSgTempRedundancyModel =
    'SA_AMF_2N_REDUNDANCY_MODEL'
  or
    s. magicCrBelongsToSgTemplate. magicCrSgTempRedundancyModel =
    'SA_AMF_NPM_REDUNDANCY_MODEL')
  then
    ceil(SIperSG/ s. magicCrBelongsToSgTemplate.
    magicCrSgTempNumberofStdbSus)
  elseif
    (s. magicCrBelongsToSgTemplate. magicCrSgTempRedundancyModel =
    'SA_AMF_NO_REDUNDANCY_MODEL')
  then
    0

```

```

endif
)
}

```

Table 6-2 The list of additional attributes

Attribute Name	Parent Element	Type	Multiplicity	Description
expectedSIsperSG	MagicCrSiTemplate	Integer	[1]	Specifies the number of SIs that are expected to be protected by a single SG
activeLoadperSU	MagicCrSiTemplate	Integer	[1]	Specifies the active load of SIs that an SU is capable to support
stdbLoadperSU	MagicCrSiTemplate	Integer	[1]	Specifies the standby load of SIs that an SU is capable to support

6.2 ETF Type Selection

This phase consists of selecting the appropriate elements from the ETF model, and pruning out the ones that do not satisfy the configuration requirements. The input and output artefacts of this transformation phase are instances of the same metamodels, namely the ETF and the Configuration Requirements sub-profiles. Therefore, the transformation phase generates an output model which is the refined input model. The output ETF Model contains exclusively the proper selected types, while the Configuration Requirements model in output will be enriched with the links to the selected ETF types.

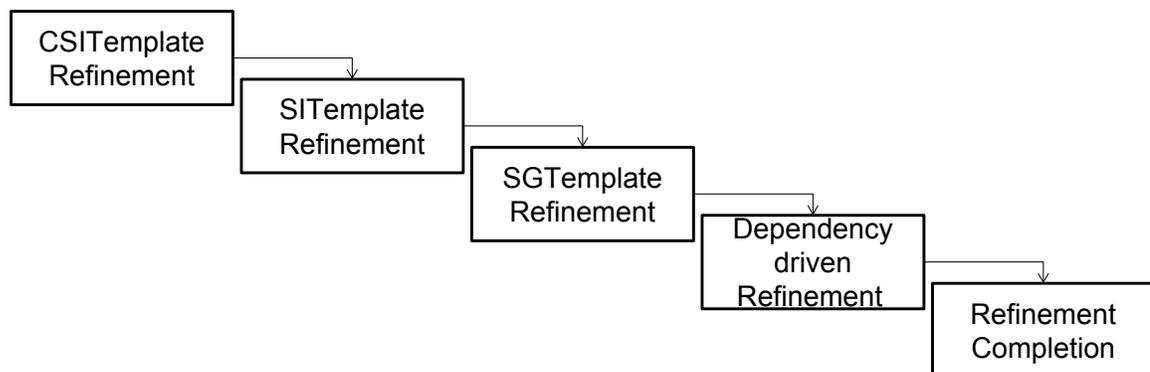


Figure 6-4 The transformation steps for ETF Type Selection phase

As shown in Figure 6-4 the type selection consists of five different steps. The first three steps bridge the gap between configuration requirements and software descriptions elements. More specifically, they establish the link between the CSITemplates, SITemplates, and SGTemplates on one end, and the appropriate ETF types to be used for the service provision on the other side. The fourth step refines the previously selected ETF types based on the dependency relationships defined at the level of configuration requirements. Finally, the fifth step aims at pruning out useless elements from the analyzed ETF model.

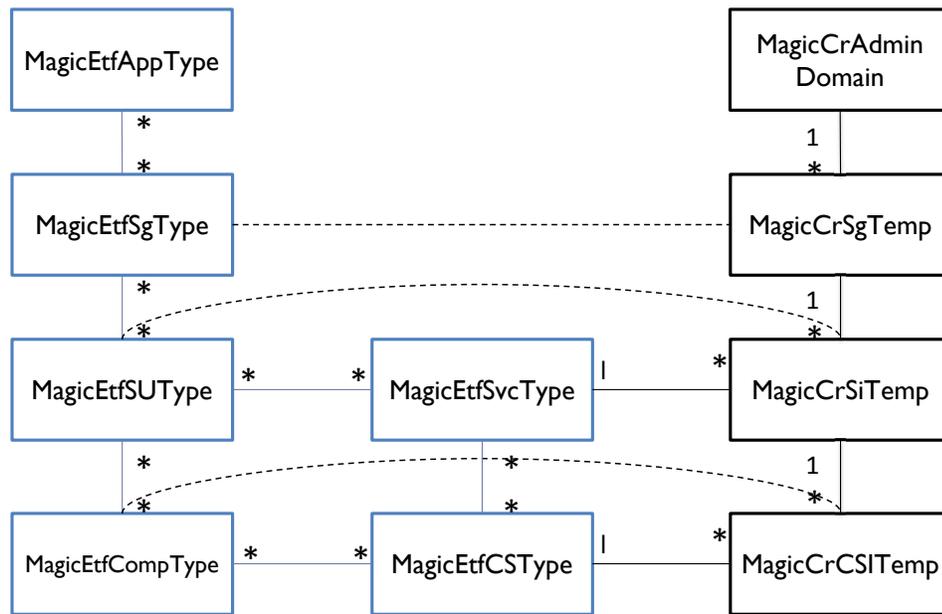


Figure 6-5 The result of the ETF Type Selection from the metamodel perspective

Figure 6-5 describes the output generated at the end of the selection phase from the metamodel perspective. The dashed connections describe the links defined between elements of the ETF and of the Configuration Requirements as the result of this phase.

6.2.1 CSITemp Refinement

The CSITemp refinement consists of selecting the Component Types capable of providing the required services described in terms of CSITemplates in the configuration requirements. The selection is operated according to different criteria:

1. The capability of providing the CSType specified by the CSITemplate.
2. The compliance of the Component Type capability model (with respect to the CSType) with the redundancy model specified by the parent SGTemplate.
3. The number of components of the Component Type that can be included in an SU and the load of assignments required to be supported by such an SU.
4. The compliance of the redundancy model specified by parent ETF SGType of the component type with the required redundancy model (specified in the parent SGTemplate).

The first two criteria are general and are required to be checked for all component types of the ETF model. The third one is checked for the component types that have at least one parent SUType in the ETF model, referred to as non-orphan component types. Moreover, if the parent SUType has at least one parent SGType in the ETF model, it is required to apply the last criterion. Figure 6-6 illustrates the refinement process using a UML activity diagram. This figure represents the control flow which regulates the usage of each selection criterion.

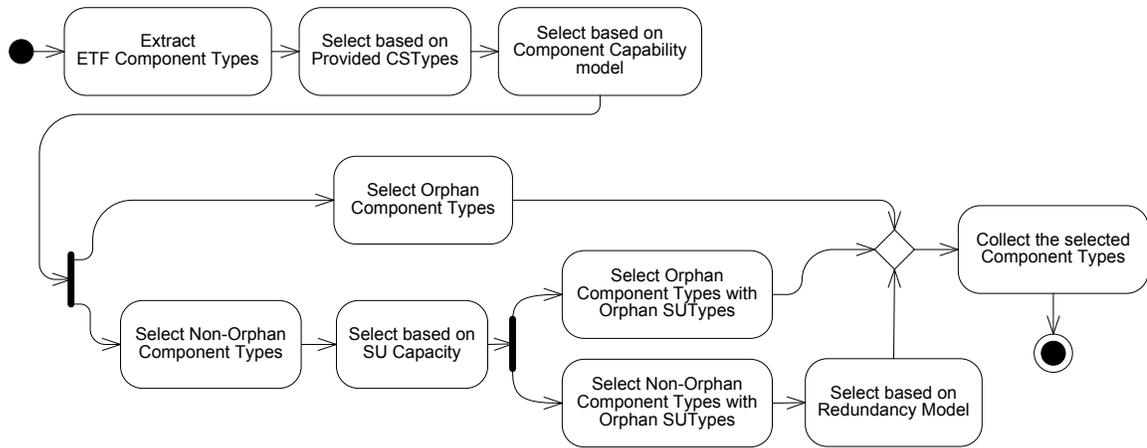


Figure 6-6 The activity diagram describing the selection of ETF Component Types

The component type selection requires visiting both input models (see Figure 6-3), with the aim to identify the proper Component Types for CSITemplates. The refinement consists of specifying the link between CSITemplates and the Component Types.

```

rule CompTypeSelection {
  from s: MagicCRProfile! MagicCrCsiTemplate
  to t: MagicCRProfile!MagicCrCsiTemplate(
    properEtfCt<- properCtFinder())}
  
```

The above code describes the transformation rule that finds the proper Component Types for each CSITemplate. The rule uses the properCtFinder helper function which implements the previously shown refinement process (see Figure 6-6). This function identifies the set of Component Types which satisfy the above mentioned criteria.

The rule fires for all instances of the CSITemplates of the configuration requirements model. The execution of this rule results in selecting the set proper ETF Component Types for each CSITemplates. However, the sets identified during this transformation step do not necessarily represent the proper set that will be used to support the generation.

As a matter of fact, they will be further refined based on additional criteria introduced in the next transformation steps.

Criterion 1: Provided CStype

Each CSITemplate specifies the CStype that identifies the type of the CSI that needs to be provided, as well as the number of CSIs. For each Component Type it is required to evaluate whether the Component Type can provide the required CStype. More specifically, this can be done comparing each required CStype with the list of CStypes that can be provided by the Component Type. The following code shows the first part of the helper function that selects the proper Component Types based on the supported CStypes. The helper defines a data structure called *selectedCompType* where it collects the selected types.

```
helper context MagicCRProfile! MagicCrCsiTemplate
def : properCtFinder() :
  Set (MagicEtfProfile!MagicEtfComptype) =
    let selectedCompType :
      Set (MagicEtfProfile!MagicEtfComptype) =
        MagicEtfCtCStype.allInstances
      -> select (ctcst|ctcst.magicEtfSupportedCsType =
        self.magicCrCsiTempCsType
      ...
```

The remaining parts of the helper function define the other selection criteria as illustrated subsequently.

Criterion 2: Component Capability Model

The component capability model of the selected Component Type must conform to the required redundancy model. The capability model specifies the capacity of the Component Type in terms of the number of active and/or standby CSI assignments (of the given CStype) that a component of that type can support. As specified in AMF sub-

profile, applying different redundancy models imposes different constraints on the capability model. The redundancy model is specified by the SGTemplate.

The following code, extracted from the helper function, expresses the constraint imposed by N-Way redundancy model.

```
...
and if self.magicCrBelongsToSiTemplate.
    magicCrBelongsToSgTemplate.magicCrSgTempRedundancyModel =
    'SA_AMF_N_WAY_REDUNDANCY_MODEL'
then
    ctst.magicEtfCompCapabilityModel =
    'MAGIC ETF_COMP_X_ACTIVE_AND_Y_STANDBY'
    ...
```

Criterion 3: Number of supported components by the SUType and SU Capacity

If the selected Component Types has a parent SUType it is required to take into consideration the number of components of the Component Type that can be included in an SU. More specifically, the number of Components of this Component Type in an SU has to be capable of supporting the load of CSIs of the particular CStype.

The load of active/standby assignments required by the CSITemplate is related to the one of the parent SITemplate. The number of SI assignments that should be supported by a SU that aggregates Components of the selected Component Types depends on the redundancy model specified in the Configuration Requirements model. The maximum load of CSIs that should be supported by such an SU is the product of the SI load and the number of CSIs specified by the current CSITemplate.

The required services need to be provided by the software entities. Therefore, it is necessary to check the capacity of Component Types and SUTypes with respect to the number of possible active/standby assignments they can provide. More specifically, we

need to find the maximum number CSIs of a CStype that can be provided by the Components aggregated in an SU. The ETF specifies the maximum number of components of a particular Component Type that can be aggregated into the SUs of a given SUType (`magicEtfMaxNumInstances`). Besides, for each Component Type, the ETF specifies also the maximum number of CSIs active/standby assignments of each supported CStype (`magicEtfMaxNumActiveCsi` and `magicEtfMaxNumStandbyCsi`). As a result, the active/standby capacity of SUs of a given SUType in handling assignments of CSIs of a given CStype is the product of `magicEtfMaxNumInstances` and `magicEtfMaxNumActiveCsi/ magicEtfMaxNumStandbyCsi`.

As a consequence, a Component Type aggregated into a given SUType can be selected only if its provided capacity can handle the load associated with the CStype of the CSITemplate.

The following ATL code extracted from *properCtFinder* selects Component Types capable of supporting the required active and standby load.

```

...
and if ctst.magicEtfSupportedby.MagicEtfCtSut->notEmpty()
  then
ctst.magicEtfSupportedby.MagicEtfCtSut
->select(ctsut|ctsut.magicEtfGroupedBy.magicEtfSvctSut
->exists(svcsut| svcsut.magicEtfProvidesSvcType =
  self.magicCrBelongsToSiTemplate.magicCrSiTempSvcType))
  ->forall(ctsutTemp|
    ctsutTemp.magicEtfMinNumInstances *
    ctst.magicEtfMaxNumActiveCsi >=
self.magicCrBelongsToSiTemplate.magicCrBelongsToSgTemplate.magicCrGroup
sSiTemplates
->collect(sitemp|sitemp.magicCrSiTempGroups)
->select(csitemp|csitemp.magicCrCsiTempCsType =
self.magicCrCsiTempCsType)
->iterate(v, active:Integer = 0| active +
v.magicCrCsiTempNumberofCsis*v.magicCrBelongsToSiTemplate.activeLoadper
SU)

```

```

and
    ctsutTemp.magicEtfMinNumInstances *
    ctst.magicEtfMaxNumStandbyCsi >=
self.magicCrBelongsToSiTemplate.magicCrBelongsToSgTemplate.magicCrGroup
sSiTemplates
->collect(sitemp|sitemp.magicCrSiTempGroups)
->select(csitemp|csitemp.magicCrCsiTempCsType =
self.magicCrCsiTempCsType)
->iterate(v, standby:Integer = 0| standby +
v.magicCrCsiTempNumberOfCsis*v.magicCrBelongsToSiTemplate.stdbLoadperSU
)
...

```

Notice that the calculation of the load is based on the activeLoadperSU/stdbLoadperSU attributes of the SiTemplates which aggregate the CiSiTemplates that require the same CsType, as well as the number of the CSIs of these CSITemplates.

Criterion 4: Redundancy model

If the parent SUType of the Component Type has a parent SGType, the redundancy model of the SGType has to match the one specified in the SGTemplate which contains the current CSITemplate. The following ATL code (part of *properCtFinder*) verifies the compliance of the redundancy model specified in the parent SGTemplate.

```

...
and
if ctsutTemp.magicEtfGroupedBy.magicEtfGroupedBy->notEmpty()
then
ctsutTemp.magicEtfGroupedBy.magicEtfGroupedBy
->exists(sgt| sgt. magicEtfSgtRedundancyModel =
self. magicCrBelongsToSiTemplate.magicCrBelongsToSgTemplate.
magicCrSgTempRedundancyModel
-- End of the properCtFinder helper function

```

At the end of this step and after considering all above mentioned criteria, if the set of Component Types selected is an empty set, the analyzed ETF model cannot satisfy the configuration requirements and therefore the configuration cannot be designed. Otherwise, the refinement process moves the focus from the level of selecting

Component Types for CSITemplates, to finding the proper SUTypes for SITemplates referred to as SITemplate refinement.

6.2.2 SITemp Refinement

The SITemp refinement consists of selecting the SUTypes of the ETF model capable of providing the services required by the SITemplates specified in the Configuration Requirements model. The selection process in this step is similar to the one defined in the CSITemp refinement. In this step the ETF model is further refined with respect to the properties required by the SITemplates and base on the following criteria:

1. The capability of providing the SvcType specified by the SITemplates aggregated by the SGTemplate of the current SITemplate.
2. The compliance of the redundancy model specified by parent ETF SGType of the SUType with the required redundancy model of SITemplate (specified in the parent SGTemplate).
3. The existence of links (resulting from the CSITemp refinement) between Component Types of the SUType and CSITemplates of the SITemplate.

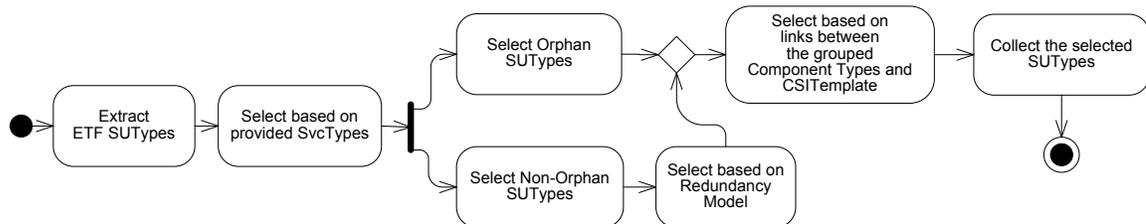


Figure 6-7 The activity diagram describing the selection of ETF SUTypes

The UML activity diagram in Figure 6-7 represents the process to select SUTypes based on these mentioned criteria.

```

rule SUTypeSelection {
from s: MagicCRProfile! MagicCrRegularSiTemplate
to t: MagicCRProfile! MagicCrRegularSiTemplate(
    properEtfSUT<- properSUTFinder())}

```

The rule, which is presented above, defines the link between the SITemplates and the selected SUTypes by using the *properSUTFinder* helper function which implements the previously mentioned criteria.

Criterion 1: Provided SvcType

Each SITemplate specifies the SvcType that identifies the type of the SIs that needs to be provided, as well as the number of SIs. For each SUType we need to evaluate whether the SUType can provide the required SvcType of the SITemplates of the parent SGTemplate. More specifically, this can be done comparing SvcTypes with the list of SvcTypes that can be provided by the SUType. The following code shows the part of the helper function that selects the proper SUTypes based on the supported SvcTypes.

```

helper context MagicCRProfile! MagicCrRegularSiTemplate
def : properSUTFinder() :
Set (MagicEtfProfile!MagicEtfSUType) =
let selectedSUType :
Set (MagicEtfProfile!MagicEtfSUType) =
    MagicEtfSvctSut.allInstances
    -> select(sutsvct| sutsvct. magicEtfProvidesSvcType =
        self.magicCrSiTempSvcType
    ...

```

Criterion 2: Redundancy Model

If the SUType has a parent SGType, the redundancy model of the SGType has to match the one specified in the SGTemplate which contains the current SITemplate. The following ATL code is (part of *properSUTFinder*) verifies the compliance of the redundancy model specified in the parent SGTemplate.

```

...
and
if sutsvct.magicEtfProvidingSuType.magicEtfGroupedBy-> notEmpty() then
sutsvct.magicEtfProvidingSuType.magicEtfGroupedBy
->exists(sgt| sgt. magicEtfSgtRedundancyModel =
self.belongsToSgTemplate.magicCRSgTempRedundancyModel)
...

```

Criterion 3: Links of grouped Component Types

In order to select an SUType for an SITemplate, the SUType should group all the Component Types which are required by the CSITemplates of the given SITemplate. In other words, for each of the CSITemplates of the SITemplate at least one of the Component Types of the SUType must have the link to that CSITemplate.

```

and
self.siTempGroups->forAll(csitemp|csitemp.properEtfCt->
intersection(sutsvct.magicEtfProvidingSuType.magicEtfGroups)-
>notEmpty())

```

6.2.3 SGTemp Refinement

The SGTemp refinement consists of selecting the SGTYPES of the ETF model capable of providing the services required by the SGTemplates specified in the Configuration Requirements model. The selection is based on the following criteria:

1. The compliance of the redundancy model specified by ETF SGTYPES with the required redundancy model in SGTemplate.
2. At least one SUType of the SGTYPES has to provide all the SvcTypes associated with the SITemplates grouped in the SGTemplate.

The UML activity diagram in Figure 6-8 represents the process to select SGTYPES base on these mentioned criteria.

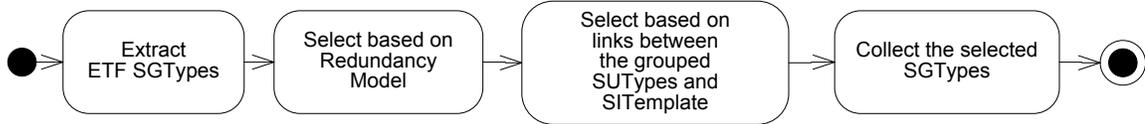


Figure 6-8 The activity diagram describing the process of selecting ETF SGTYPES

```

rule SGTypeSelection {
from s: MagicCRProfile! MagicCrSgTemplate
to t: MagicCRProfile! MagicCrSgTemplate (
    properEtfSGT<- properSGTFinder())}
  
```

Based on these criteria the *SGTypeSelection* defines the link between the SGTemplates and the selected SGTYPES. It invokes the *properSGTFinder* helper function which follows the process specified in Figure 6-8.

Criterion 1: Redundancy Model

In order to select an SGType for an SGTemplate, the SGType, the redundancy model of the SGType has to match the one specified in the SGTemplate. The following ATL code (part of *properSGTFinder*) verifies the compliance of the redundancy model specified in the parent SGTemplate.

```

helper context MagicCRProfile!SGTemplate
def : properSGTFinder() :
  Set (MagicEtfProfile!MagicEtfSGType) =
  let selectedSGType :
    Set (MagicEtfProfile!MagicEtfSGType) =
      MagicEtfSGType.allInstances
      -> select(sgt| sgt.magicEtfSgtRedundancyModel =
        self.magicCrSgTempRedundancyModel
    ...
  
```

Criterion 2: Links of grouped SUTypes

In order to select an SGType for an SGTemplate, the SGType should group at least one SUType which is required by all the SITemplates of the given SGTemplate. In other

words, this SUType is capable of providing each of the SvcType associated with the SITemplats aggregated in the SGTemplate.

```
...
and
sgt.saAmfSgtValidSuTypes -> exists(sut| sut->
magicSaAmfSutProvidesSvcType
->includesAll(self.magicCrGroupsSiTemplates
->collect(sit|sit.magicCrSiTempSvcType))
```

6.2.4 Dependency Driven Refinement

In this step, we take into account the dependency relationships that exist both at the level of configuration requirements elements and at ETF model elements level. In the configuration requirements model the dependency relationships are defined between CSITemplates and between SITemplates. In the ETF model, the dependency relationships are specified between the Component Types in providing CSTypes and between SUTypes in providing SvcTypes. The objective of this step is to refine the previously selected ETF types based on the dependency relationships defined at the level of configuration requirements. More specifically, all ETF types that do not respect the dependency requirements need to be pruned out form the set of selected types.

The refinement consists of two different activities: 1) refinement of the set of proper Component Types for each CSITemplate, 2) refinement of the set of appropriate SUTypes for each SITemplate.

6.2.4.1 Component Type Dependency driven Refinement

This activity aims at refining the set of Component Types selected as a result of previous step based on the dependency relationships. The refined set of Component Types needs to be compliant with the configuration requirements from the dependency point of view. To

this end, this refinement activity takes into account the following scenario for each CSITemplate: In case the CSITemplate does not specify any dependency relationship to other CSITemplates, the proper Component Types for the CSITempalte should not have any dependency in providing the required CStype.

This activity is described in terms of a refinement transformation of CSITemplates. The transformation is enabled for each CSITemplate in the Configuration Requirements model which does not specify any dependency relationship.

The refinement consists of updating the set of properCt by including in this set only those Component Types that do not specify any dependency in providing the CStype associated with the CSITemplate. This refinement takes into account the dependency relationship in both directions. More specifically, it considers both the case in which a CompType depends on other CompTypes in providing a CStype, and the case in which a given CompType in providing a CStype depend by other CompTypes.

These two cases are implemented in terms of two different ATL rules. The first rule, *CSITempNotDependOnRefinement*, extracts from the set of previously selected Component Types of a given CSITemplate those that do not depend on any other component types in providing the associated CStype.

```
rule CSITempNotDependOnRefinement {
  from
    s: MagicCRProfile!MagicCrCsiTemplate
      s.magicCrCsiTempDependsOn->IsEmpty()
    )
  to
    t: MagicCRProfile!MagicCrCsiTemplate(
      properEtfCt<-
        s.magicCrCsiTempCsType.MagicEtfCtCStype->
      select (sourceCst|s.properEtfCt->includes(
        sourceCst.magicEtfSupportedby))->select(
```

```

        targetctcst| targetctcst.magicEtfRequires->IsEmpty())
    )->collect(ctcst|ctcst.magicEtfSupportedby)
)
}

```

The second rule, named *CSITempNotDependByRefinement*, refines the set of proper Component Types of a given CSITemplate by selecting the ones that do not depend on by any other Component Types.

```

rule CSITempNotDependByRefinement {
from
    s: MagicCRProfile!MagicCrCsiTemplate(
        s.magicCrCsiTempDependedOnBy->IsEmpty()
    )
to
    t: MagicCRProfile!MagicCrCsiTemplate(
        properEtfCt<-
        s.magicCrCsiTempCsType.MagicEtfCtCsType->
select(sourcectcst|s.properEtfCt->includes(
sourcectcst.magicEtfSupportedby))->select(
        targetctcst| targetctcst.magicEtfRequiredBy->IsEmpty())
    )->collect(ctcst|ctcst.magicEtfSupportedby)
)
}

```

6.2.4.2 *SUType Dependency driven Refinement*

This activity aims at refining the set of SUTypes selected as a result of previous step based on the dependency relationships. The refined set of SUTypes needs to be compliant with the configuration requirements from the dependency point of view. To this end, this refinement activity takes into account the following scenario for each SITemplate: In case the SITemplate does not specify any dependency relationship to other SITemplates, the proper SUTypes for the SITemplate should not have any dependency in providing the required SvcType.

This activity is the refinement transformation of SITemplates. The transformation is fired for each SITemplate which does not specify any dependency relationship.

The refinement consists of selecting from the set of properSUT the SUTypes that do not specify any dependency on other SvcTypes in providing the SvcType associated with the SITemplate. This transformation is implemented using the ATL rule *SITempNotDependOnRefinement*.

```

rule SITempNotDependOnRefinement {
from
    s: MagicCRProfile! MagicCrRegularSiTemplate (
        s.magicCrSiTempDependsOn->IsEmpty()
    )
to
    t: MagicCRProfile! MagicCrRegularSiTemplate (
        properEtfSUT<-
        s.magicCrSiTempSvcType.MagicEtfSvctSut-> select (
            sourcesutsvct|s.properEtfSUT>includes(sourcesutsvct.magicEtfProvidin
            gSuType)
        )->select(targetsutsvct| targetsutsvct.magicEtfRequires->IsEmpty())

        )->collect(sutsvct|sutsvct.magicEtfProvidingSuType)
    )
}

```

6.2.5 Completing the Refinement

The previously selected ETF types represent the essential software resources that can be used to design an AMF configuration which satisfies the configuration requirements. As previously mentioned, the proper sets identified at the end of each selection step need to be further refined since they may contain elements which are inappropriate to be used for generation purposes. More specifically, the previously mentioned criteria consider each selected ETF type as independent from the other ETF types. For example, a selected ETF Component Type is aggregated by an ETF SUType which has not been selected during the SUType refinement step. That Component Type cannot be used for generation purposes and thus has to be removed from the selected sets. This transformation phase is completed pruning out the unselected irrelevant types from the ETF model. This

refinement activity results in the sets of ETF types that will be used for the subsequent phases of the transformation. Figure 6-9 illustrates the different activities that characterize the completion of the refinement.

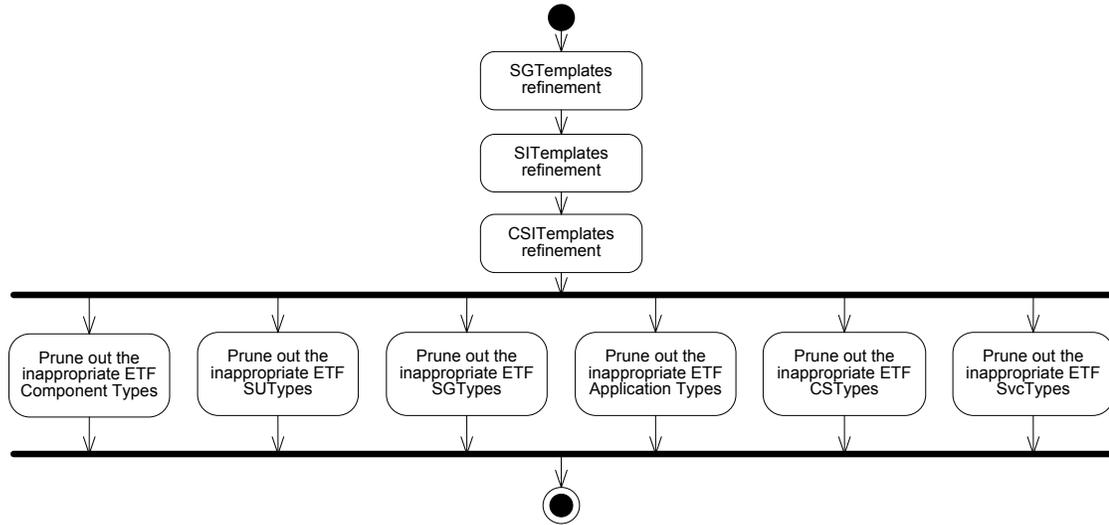


Figure 6-9 The transformations performed to complete the refinement phase

More specifically, the transformation starts refining the selected set of ETF types linked by each Configuration Requirements element. Afterwards, it forwards the appropriate ETF types to the next phase pruning out the unselected ones from the ETF model.

6.2.5.1 Configuration requirements refinement

In this step the CR model is transformed refining the list of proper ETF types based on different criteria. The step is characterized by three different transformations.

The SGTemplate elements and their previously defined links to the ETF types are forwarded to next phase of the transformation without any change as specified in following ATL code.

```

rule SgtempRefinement {
from s: MagicCRProfile! MagicCrSgTemplate
to t: MagicCRProfile!MagicCrSgTemplate
}

```

The SITemplate elements are refined modifying the associated list of proper SUTypes by means of SItempRefinement transformation rule. This rule prunes the irrelevant SUTypes from the preliminary selected set. More specifically an SUType will result in the final set of selected SUTypes:

- If it is not aggregated by any SGType
- If it is aggregated by an SGType and the SGType is in the set of selected SGTypes of the SGTemplate associated with the current SITemplate.

```

rule SItempRefinement {
from s: MagicCRProfile! MagicCrRegularSiTemplate
to t: MagicCRProfile! MagicCrRegularSiTemplate(
properEtfSUT<- s.properEtfSUT->select(sut| sut.magicEtfGroupedBy-
>IsEmpty() or
sut.magicEtfGroupedBy -> intersection(self.magicCrBelongsToSgTemplate->
properEtfSGT)-> notEmpty()))
}

```

Afterwards, CSITemplate elements are transformed updating the list of proper Component Types. The inappropriate Component Types are pruned out from the list based on criteria similar to the ones used for SITemplate. More specifically a Component Type will result in the proper selected set:

- If it is not aggregated by any SUType
- If it is aggregated by an SUType and the SUType is in the set of selected SUTypes of the SITemplate associated with the current CSITemplate.

```

rule CSItempRefinement {
from s: MagicCRProfile! MagicCrCsiTemplate
to t: MagicCRProfile! MagicCrCsiTemplate(
  properEtfCt<- s.properEtfCt->select(ct| ct.magicEtfGroupedBy->
    isEmpty() or
ct.magicEtfGroupedBy -> intersection(self.magicCrBelongsToSiTemplate->
properEtfSUT)->notEmpty()))
}

```

6.2.5.2 ETF type refinement

In this step, the ETF model is transformed pruning out the inappropriate ETF types from the current ETF model. The refinement is operated based on the previously selected and refined set of proper ETF types linked by the configuration requirements elements.

Component Types pruning

The transformation consists of creating into the target model, Component Type elements with the same set of attributes of the selected ones. The following code focuses on the Component Type set that has been previously linked to the CSITemplates.

```

rule CompTypePruning {
from s: MagicETFProfile!MagicEtfCompType
(MagicCRProfile::MagicCrCsiTemplate.allInstances-> exists(csitemp|
csitemp.properEtfCt->includes(s)))
to t: MagicETFProfile!MagicEtfCompType(
  magicEtfCtVersion<- s.magicEtfCtVersion
  -- Transforming the rest of the attributes.....
)
}

```

SUType pruning

This transformation prunes the irrelevant SUTypes based on the preliminary selection performed during the first refinement step and the relationships with the ETF SGType resulting from the previously described selection steps. More specifically an SUType will result in the final set of selected SUTypes:

- If it is not aggregated by any SGType

- if it is aggregated by an SGType, the SGType should be in the set of selected SGTypes of the SGTemplate associated with its SITemplates

The transformation consists of copying the selected SUType elements into the target model.

```
rule SUTypePruning {
from s: MagicETFProfile!MagicEtfSUType
(MagicCRProfile:: MagicCrRegularSiTemplate.allInstances->forall(sitemp |
sitemp.properEtfSUT->includes(s)))
to t: MagicETFProfile!MagicEtfSUType(
magicEtfSutVersion<- s.magicEtfSutVersion
-- Transforming the rest of the attributes.....
```

SGType pruning

Similar to the previous pruning steps the transformation consists of replicating the selected SGType elements into the target model.

```
rule SGTypePruning {
from s: MagicETFProfile!MagicEtfSGType
(MagicCRProfile:: MagicCrSgTemplate.allInstances->exists(sgtemp |
sgtemp.properEtfSGT->includes(s)))
to t: MagicETFProfile!MagicEtfSGType(
magicEtfSgtVersion<- s.magicEtfSgtVersion
-- Transforming the rest of the attributes.....
```

Application Type pruning

A similar rule can be applied for the pruning out the Application Types. However, this pruning also requires identifying the Application Types capable of supporting at least one of the previously selected SGTypes. In fact, there is no element in the configuration requirement model that directly links to the Application Types.

```
rule APTypePruning {
from s: MagicETFProfile!MagicEtfAppType
(MagicCRProfile:: MagicCrSgTemplate.allInstances
->exists(sgtemp | sgtemp.properEtfSGT
->intersection(s.magicEtfGroups)->notEmpty()))
to t: MagicETFProfile!MagicEtfAppType(
magicEtfApptVersion <-s. magicEtfApptVersion
-- Transforming the rest of the attributes.....
```

SvcType pruning

The SvcTypes pruning is easily realized by operating on the SvcTypes which are linked by SITemplates.

```
rule SvcTypePruning {
from s: MagicETFProfile!MagicEtfSvcType
(MagicCRProfile::SITemplate.allInstances
->exists(sitemp| sitemp. magicCrSiTempSvcType = s)
to t: MagicETFProfile!MagicEtfSvcType (
    magicEtfSvctVersion<-s. magicEtfSvctVersion
    -- Transforming the rest of the attributes.....
```

CSType pruning

The CSTypes pruning is easily realized by operating on the CSTypes which are linked by CSITemplates.

```
rule CSTypePruning {
from s: MagicETFProfile!MagicEtfCSType
(MagicCRProfile::CSITemplate.allInstances
->exists(csitemp| csitemp. magicCrCsiTempCsType = s)
to t: MagicETFProfile!MagicEtfCSType (
    magicEtfCstVersion<-s. magicEtfCstVersion
    -- Transforming the rest of the attributes.....
```

6.3 AMF Entity Type Creation

This phase mainly consists of generating the AMF entity types to be used for the AMF configuration design. The main objective of this phase is to define the AMF entity types that can be used to specify one possible configuration which satisfies the configuration requirements.

As shown in Figure 6-3, this transformation phase takes as input the ETF model refined by the previous transformation phase described in 6.2. This phase creates and configures AMF entity types based on the selected ETF types. It also creates the links between AMF entity types and Configuration Requirements considering the possible relationships that

exists between the ETF types and CSITemplates, SITemplates, or SGTemplates. More specifically, these links substitute the links between ETF types and templates resulting from the previous phase. For example, an AMF Component Type can be created based on a selected ETF Component Type in the refined ETF model. In addition the generated AMF Component Type is linked to the CSITemplates which is already connected to the ETF type.

Figure 6-10 describes the output generated at the end of this phase from the metamodel perspective. The dashed connections describe the links defined between the generated AMF entity types and the elements of Configuration Requirements as well as the relationships among the AMF entity types.

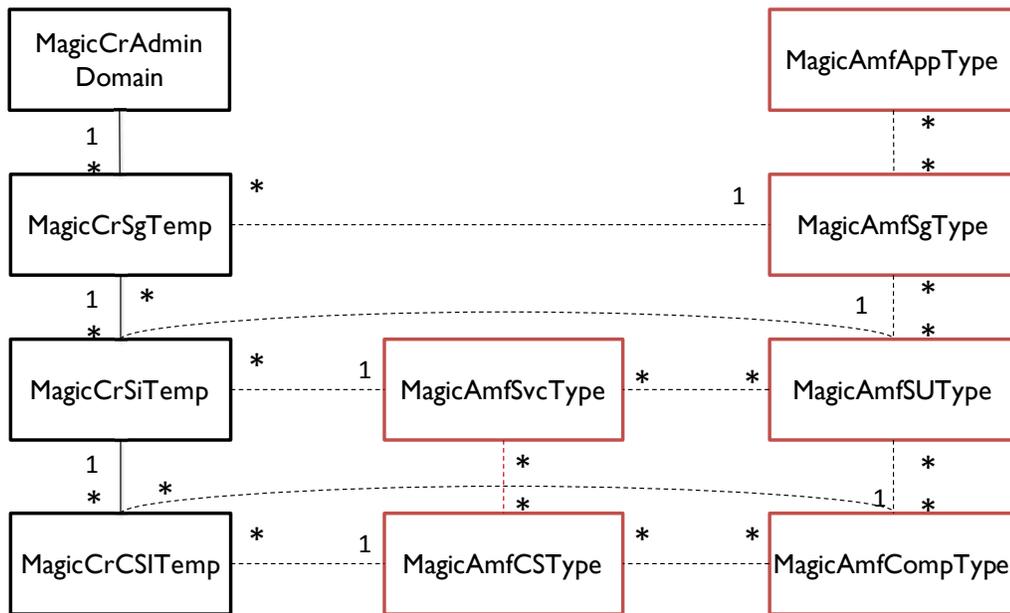


Figure 6-10 The result of the AMF Entity Type creation phase from the metamodel perspective

Figure 6-11 presents the AMF entity type creation phase as composed of four different steps.

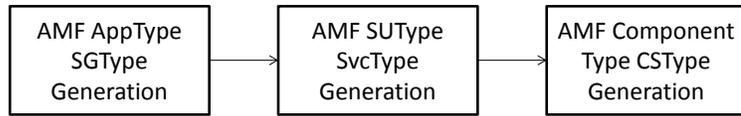


Figure 6-11 The transformation steps of the AMF entity type creation phase

Each step corresponds to a different transformation that generates a particular AMF entity types starting from the corresponding previously selected ETF types. However, the only mandatory elements in ETF model are Component Types and CTypes. Therefore, SUTypes, SGTYPES, AppTypes and SvcTypes might not exist in the ETF model. The refinement phase described in the previous section does not aim at modifying the ETF model by completing the definition of the missing ETF types. In other words, it is possible to have ETF types that are not aggregated into other ETF types according to the hierarchical structure specified by the ETF model. For example, ETF Component Types may not be aggregated by any ETF SUType. Although missing types are tolerated in ETF models, in order to generate an AMF configuration it is required to have the complete hierarchy of types. Therefore, to complete the hierarchy, the transformation process builds AMF entity types based on a set of existing ETF types. For the previously mentioned example, we need to create an AMF SUType based on the existing ETF Component Types.

In this section, we discuss the details of the AMF entity type creation phase and present the transformation rules accordingly. In this phase we start generating the different AMF entity types directly derived from existing ETF types, and afterwards, we focus on creating the AMF entity types which do not have any ETF type counterpart. Besides generating the proper AMF types, these transformations also establish the required relationships among them.

For the creation of the AMF entity types based on the existing ETF types the generated AMF entity types are characterized by a set of attributes that directly corresponds to the properties defined in ETF types. As a matter of fact, the properties specified in ETF types impose restrictions on corresponding AMF entity types' attributes. For instance, they can specify the admissible range of values that can be defined for each attribute. For the sake of simplicity, the same values defined in ETF types are assigned to these attributes. In case of optional attributes which are not specified in the ETF model, for the entity type generation we create them without any initial value.

In order to generate AMF entity types that do not have any ETF counterparts, these generated AMF entity types are characterized by a set of attributes which are initialized with the information described in configuration requirement elements (e.g. redundancy model which is specified in the SGTemplate). Moreover, in case we have attributes without any value, in our approach we initialize them according the default values indicated in the AMF specification.

6.3.1 AMF SGType and AppType Generation

As previously mentioned, SGTypes and Application Types are not mandatory elements of an ETF models. Moreover, there is no element in the configuration requirement model that directly links to the Application Types. Therefore, the generation of both AMF SGTypes and AppTypes will be performed starting from SGTemplate and based on the set of selected SGTypes of that template. The generation is implemented using three different transformations:

1. If the list of selected ETF SGTYPES is empty, we need to create an AMF SGTYPe and a parent AMF AppType from scratch.
2. If the list of selected ETF SGTYPES consists of only orphan SGTYPES, we transform one of the selected ETF SGTYPES and create the parent AMF AppType from scratch.
3. If the list of selected ETF SGTYPES consists of at least one non-orphan SGTYPe, we transform one of the non-orphan SGTYPES and one of its parent AppTypes.

The first transformation is rather straightforward and performed by means of a single ATL rule called *AMFSGType_AppTypeCreate*. For a given SGTemplate, this rule fires if the list of the proper ETF SGTYPES is empty, indicating that there is not an appropriate SGTYPe in the ETF model for this SGTemplate. The rule consists of three different parts: *t1* creates an AMF SGTYPe and *t2* generates the AMF AppType. Moreover, the link between the created AMF SGTYPe and AppType is also established in *t1* and *t2*. Finally, *t3* creates the link to the generated AMF SGTYPe.

```

rule AMFSGType_AppTypeCreate {
from
  s: MagicCRProfile!MagicCrSgTemplate(
    --Fire only if the list of selected SGTYPES is empty
    properEtfSGT->IsEmpty()
  )
to
  t1: MagicAmfProfile! MagicAmfSGType (
    --Link to AppType
    magicSaAmfSgtMemberOf<- Set{t2},
    --Transforming the Attributes
  ),

  t2: MagicAmfProfile! MagicAmfAppType (
    --Link to SGTYPe
    magicAmfApptSGTYPES <- Set{t1},
    --Transforming the Attributes
  ),
  t3: MagicCRProfile!MagicCrSgTemplate(
    properAmfSGT <- Set{t1},
  )
}

```

The second transformation is performed by means of *Orphan_AMFSGType_AppTypeCreate* rule and *AMFSGTypeTransform* unique lazy rule. For a given SGTemplate, *Orphan_AMFSGType_AppTypeCreate* fires when all selected ETF SGTypes are orphans. In this rule *t1* creates an AMF AppType and generates the link to an AMF SGType transformed from the first selected ETF SGType. This ETF SGType is transformed using the *AMFSGTypeTransform* rule. Moreover, *t2* updates the SGTemplate with the newly created AMF SGType. *t2* also replaces the list of proper ETF SGTypes with the ETF SGType which is transformed to the AMF SGType (This list will be used in the next steps).

```

rule Orphan_AMFSGType_AppTypeCreate {
from
  s: MagicCRProfile!MagicCrSgTemplate(
    --Fire only if all selected SGTypes are orphan
    properEtfSGT->forall(sgt| sgt. magicSaAmfSgtMemberOf->IsEmpty())
  )
to

  t1: MagicAmfProfile! MagicAmfAppType (
    --Link to SGType
    magicAmfApptSGTypes <- Set{ AMFSGTypeTransform (s.properEtfSGT-
>at(1))},
    --Transforming the Attributes
  ),
  t2: MagicCRProfile!MagicCrSgTemplate(
    properAmfSGT <- Set{ AMFSGTypeTransform (s.properEtfSGT-
>at(1))},
    properEtfSGT <- Set{ s.properEtfSGT->at(1) }
  )
}

unique lazy rule AMFSGTypeTransform {
from
  s: MagicEtfProfile! MagicEtfSGType
to
  t: MagicAMFProfile! MagicAmfSGType(
    --Transforming the Attributes
  )
}

```

The third transformation fires if at least one of the selected ETF SGTYPES of a given SGTemplate is non-orphan. The transformation is performed mainly by an ATL rule called *Non_Orphan_AMFSGType_AppTypeCreate*. This rule transforms one of the non-orphan selected ETF SGTYPES to an AMF SGTYPES using *AMFSGTypeTransform* unique lazy rule and creates a link from the SGTemplate to the created AMF SGType. It also replaces the list of proper ETF SGTYPES with the ETF SGType which is transformed to the AMF SGType (This list will be used in the next steps). Moreover, by using *AMFAppTypeTransform* unique lazy rule, *Non_Orphan_AMFSGType_AppTypeCreate* transforms one of the ETF AppTypes that aggregates the transformed ETF SGType to an AMF AppType.

```

rule Non_Orphan_AMFSGType_AppTypeCreate {
from
    s: MagicCRProfile!MagicCrSgTemplate(
        --Fire only if at least one of selected SGTYPES is non-orphan
        not properEtfSGT->forall(sgt| sgt. magicSaAmfSgtMemberOf-
>IsEmpty())
    )
to

    t: MagicCRProfile!MagicCrSgTemplate(

properAmfSGT <- Set{ AMFSGTypeTransform (s.properEtfSGT->
    select(sgt|sgt. magicEtfSgtGroupedBy->notEmpty())->at(1)},
properEtfSGT <- Set{ s.properEtfSGT->select(sgt|sgt.
magicEtfSgtGroupedBy
-> notEmpty())->at(1)}

do
{
t. properAmfSGT->at(1). magicSaAmfSgtMemberOf <- AMFAppTypeTransform(t.
properAmfSGT-> at(1).magicEtfSgtGroupedBy->at(1));
}

unique lazy rule AMFSGTypeTransform {
from
    s: MagicEtfProfile! MagicEtfSGType
to
    t: MagicAMFProfile! MagicAmfSGType(
        --Transforming the Attributes
    )
}

```

```

unique lazy rule AMFAppTypeTransform {
from
    s: MagicEtfProfile! MagicEtfAppType
to
    t: MagicAMFProfile! MagicAmfAppType(
        --Transforming the Attributes
    )
}

```

6.3.2 AMF SUType and SvcType Generation

Similar to SGTypes and AppTypes, SUTypes and SvcTypes are not mandatory elements of an ETF models. However, since we assumed that the Configuration Requirements model is complete, the SvcTypes are already specified in this model. Therefore, different generation strategies need to be defined according to the existence of the SUTypes in the ETF model. As a consequence, this generation step consists of three different transformations.

1. Generation of the AMF SUTypes and SvcTypes from the selected matching non-orphan ETF SUTypes and the related ETF SvcType.
2. Generation of the AMF SUTypes and SvcTypes from the selected matching orphan ETF SUType and the related ETF SvcType.
3. Creation of the AMF SUTypes from scratch as well as the creation of the AMF SvcTypes based on the corresponding ETF types. This transformation covers the case in which the corresponding ETF SUTypes are missing in the selected ETF model.

In the rest of this section we describe the details of the above mentioned transformations using ATL.

The first transformation generates AMF SvcTypes and AMF SUTypes for a given SITemplate starting from the corresponding ETF SvcType and non-orphan ETF SUTypes selected in the previous step.

The transformation is implemented using the *Non_Orphan_AMFSUType_SvctTransform* rule and the unique lazy rules *AMFSUTypeTransform* and *AMFSvctTransform*. The rule also establishes the relationships among the generated AMF types. Moreover, the lazy rules generate AMF types that capture all the characteristics of the related ETF types. The rule also establishes the relationships among the generated AMF types.

For a given SITemplate, *Non_Orphan_AMFSUType_SvctTransform* refines the current SITemplate and targets the generation of AMF SvcTypes and SUTypes and their relationships. The rule fires only if any of the selected SUTypes is not an orphan. Thereafter, the rule transforms the ETF SUType which is supported by the SGType transformed for the aggregating SGTemplate in previous step. This transformation is performed by calling the *AMFSUTypeTransform*. *AMFSUTypeTransform* targets the generation of AMF SUTypes and the attributes based on the corresponding ETF types. The list of proper ETF SUTypes is also replaced with the ETF SUType which is transformed to the AMF SUType (This list will be used in the next steps).

```
rule Non_Orphan_AMFSUType_SvctTransform {
  from
    s: MagicCRProfile!MagicCrRegularSiTemplate(
      --Fire only if properEtfSUT has an SUTypes which is the child of the
      transformed SGType of the aggregating SGTemplate
      s.properEtfSUT->exists(sut|s.magicCrBelongsToSgTemplate.
      properEtfSGT
      ->at(1).magicEtfGroups->includes(sut)))
  to
    t: MagicCRProfile!MagicCrRegularSiTemplate(
      --Link to SUType
```

```

properAmfSUT <- Set{ AMFSUTypeTransform(s.properEtfSUT->select(sut|s.
magicCrBelongsToSgTemplate. properEtfSGT->at(1).magicEtfGroups-
>includes(sut)))
->at(1)},
  --Link to SvcType
  magicCrSiTempSvcType<- Set{AMFSvctTransform(s.
magicCrSiTempSvcType->at(1))}
),

do {

  let svct : MagicAMFProfile!MagicSaAmfSvcType =
    AMFSvctTransform(s.magicCrSiTempSvcType->at(1))

  in

svct.magicAmfSvcTProvidingSut <- Set{ AMFSUTypeTransform(s.properEtfSUT
->select(sut|s. magicCrBelongsToSgTemplate. properEtfSGT
->at(1).magicEtfGroups->includes(sut)))->at(1)};
properEtfSUT <- Set{ s.properEtfSUT->select(sut|s.
magicCrBelongsToSgTemplate. properEtfSGT->at(1).magicEtfGroups-
>includes(sut))->at(1)}
}
}

```

Moreover, *AMFSvctTransform* generates the *SvcType* associated with the current *SITemplate* and initializes its attributes with the attributed specified by the corresponding *ETF* type.

```

unique lazy rule AMFSvctTransform{
from
  s: MagicEtfProfile!MagicEtfSvcType
to
  t: MagicAMFProfile!MagicSaAmfSvcType(
    --Transforming the Attributes
    safSvcType <- s.magicCrSiTempSvcType.magicEtfSvctName,
    magicSafVersion <- s.magicCrSiTempSvcType.magicEtfSvctVersion
  )
}

```

The second transformation generates *AMF SvcTypes* and *AMF SUTypes* for a given *SITemplate* starting from the corresponding *ETF SvcType* and orphan *ETF SUTypes* selected in the previous phase.

This transformation is implemented using the *Orphan_AMFSUType_SvctTransform* rule and the unique lazy rules *AMFSUTypeTransform* and *AMFSvctTransform* similar to the first transformation.

For a given SITemplate, *Orphan_AMFSUType_SvctTransform* refines the current SITemplate and targets the generation of AMF SvcTypes and SUTypes and their relationships. The rule fires only if all selected SUTypes are orphans. This also implies that the SGTemplate of the current SITemplate was created from scratch in the previous step. Consequently, the rule transforms the one of the selected ETF SUTypes by calling the *AMFSUTypeTransform*. *AMFSUTypeTransform* targets the generation of AMF SUTypes and the attributes based on the corresponding ETF types. The list of proper ETF SUTypes is also replaced with the ETF SUType which is transformed to the AMF SUType (This list will be used in the next steps). Finally, *AMFSvctTransform* generates the SvcType associated with the current SITemplate.

```

rule Orphan_AMFSUType_SvctTransform {
from
  s: MagicCRProfile!MagicCrRegularSiTemplate(
  --Fire only if all Component Types of properEtfSUT are orphan
    not s.properEtfSUT->exists(sut|s.magicCrBelongsToSgTemplate.
properEtfSGT
    ->at(1).magicEtfGroups->includes(sut)))
to
  t: MagicCRProfile!MagicCrRegularSiTemplate(
    --Link to SUType
    properAmfSUT <- Set{ AMFSUTypeTransform(s.properEtfSUT->
at(1)),
    --Link to SvcType
    magicCrSiTempSvcType<- Set{AMFSvctTransform(s.
magicCrSiTempSvcType->at(1)) }
  ),
do {
    let svct : MagicAMFProfile!MagicSaAmfSvcType =
      AMFSvctTransform(s.magicCrSiTempSvcType->at(1))
    in

```

```

svct.magicAmfSvcTProvidingSut <- Set{
AMFSUTypeTransform(s.properEtfSUT
->at(1));
  --Replace the properEtfSUT with the ETF SUType which is transformed
to AMF SUType
s.properEtfSUT<- Set{ s.properEtfSUT->at(1) }

  }
}

```

As mentioned the above presented rules (*Non_Orphan_AMFSUType_SvctTransform*, *Orphan_AMFSUType_SvctTransform*, *AMFSUTypeTransform*, and *AMFSvctTransform*) aim at generating AMF types from the existing corresponding ETF types. However, if the selection phase could not find any appropriate SUType for the given SITemplate, the proper SUType needs to be created from scratch.

The *AMFSUType_SvctCreate* rule creates AMF SUTypes from scratch and the AMF SvcTypes based on the corresponding ETF types.

In *AMFSUType_SvctCreate* for a given SITemplate, the firing condition checks the existence of an ETF SUType in list of selected SUTypes (properEtfSUT). The rule targets the creation of different AMF entity types and the relationship among them. For each created entity type the transformation initializes the attributes to their default value specified in AMF specification. More specifically, *t* defines the link between the SITemplates and the newly created SUTypes (from scratch) and SvcType generated by the *createSUTfromScratch* helper function and the lazy rule *AMFSvctTransform*, respectively.

```

rule AMFSUType_SvctCreate {
from
  s: MagicCRProfile!MagicCrRegularSiTemplate(
    --Fire only if the list of selected SUTypes is empty
    properEtfSUT->IsEmpty()
  )
to

```

```

t: MagicCRProfile!MagicCrRegularSiTemplate(
  --Link to SUType
  properAmfSUT <- Set{s.createSUTfromScratch()},
  --Link to SvcType
  magicCrSiTempSvcType<- Set{AMFSvctTransform(s.
magicCrSiTempSvcType->at(1))}
),
do {
  let svct : MagicAMFProfile!MagicSaAmfSvcType =
    AMFSvctTransform(s.magicCrSiTempSvcType->at(1))
  in
    svct.magicAmfSvcTProvidingSut <-
Set{s.createSUTfromScratch()}
}
}

unique lazy rule AMFSvctTransform{
from
  s: MagicEtfProfile!MagicEtfSvcType
to
  t: MagicAMFProfile!MagicSaAmfSvcType(
    --Transforming the Attributes
    safSvcType <- s.magicCrSiTempSvcType.magicEtfSvcName,
    magicSafVersion <- s.magicCrSiTempSvcType.magicEtfSvcVersion
  )
}

helper context MagicCRProfile!SiTemplate def : createSUTfromScratch() :
MagicAMFProfile!MagicSaAmfSUType = .....

```

AMFSvctTransform creates the appropriate AMF SvcType for the associated ETF type linked to the SiTemplates. *createSUT-fromScratch* is the helper function which returns the proper AMF SUType capable of providing all the SvcTypes referred by the set of SiTemplates grouped by the SGTemplate. The function checks whether an SUType with such characteristics has already been defined in the AMF model. In this case it returns this type; otherwise it creates an SUType from scratch and returns it.

Finally, *AMFSUType_SvctCreate* establishes the connection between the newly created SvcType and the SUTypes returned by the helper function.

6.3.3 AMF Component Type and CStype Generation

AMF Component Types and AMF CStypes for a given CSITemplate are generated starting from the previously selected ETF types. These generated types capture the characteristics of the referenced ETF types.

The creation targets different elements: namely, the CStype associated with the current CSITemplate, the proper ComponentTypes, the association class that links AMF Component Types to the CStypes, the association class that links AMF Component Types to the SUTypes generated in the previous step, the association class that links CStype to the SvcType of aggregating SITemplate as well as the link between CSITemplates and the created entity types. For this purpose, we define two main transformations in order to cover the following cases:

1. Generation of the AMF Component Types and CStypes from the selected matching non-orphan ETF Component Types and the related ETF CStype as well as the generation of the association classes between AMF entity types generated both in this step and in the previous step (see Section 6.3.2).
2. Generation of the AMF Component Types and CStypes from the selected matching orphan ETF Component Types and the related ETF CStype as well as the generation of the association classes between AMF entity types generated both in this step and in the previous step (see Section 6.3.2).

The generation process for both above mentioned cases is directly illustrated by means of the transformation rule *Non_Orphan_AMFCompType_CStypeTransform* and *Orphan_AMFCompType_CStypeTransform* as well as the required unique lazy rules namely *AMFCompTypeTransform* and *AMFCStypeTransform*.

Non_Orphan_AMFCompType_CSTypeTransform refines a given CSITemplate and targets the generation of AMF CTypes and Component Types and their relationship. The rule fires only if any of the selected Component Types are not orphans. Afterwards, the rule transforms the ETF Component Type which is supported by the SUType which is transformed for the aggregating SITemplate in previous step. This transformation is performed by calling the *AMFCompTypeTransform*. *AMFCompTypeTransform* targets the generation of AMF Component Types and the attributes based on the corresponding ETF types. Moreover, *AMFCSTypeTransform* generates the CType associated with the current CSITemplate and initializes its attributes with the attributed specified by the corresponding ETF type.

t1 creates the association class (MagicSaAmfCtCSType) between the AMF Component Type and CType by calling the unique lazy rules. *t2* generates the relationship (MagicSaAmfSvcTypeCSType) between the newly created AMF CType and its parent SvcType created in the previous step. *t3* establishes the link (MagicSaAmfSutCompType) between the newly generated AMF Component Type and its parent SUType created in the previous step. Finally, *t4* updates the CSITemplate with the list of AMF Component Types.

```

rule Non_Orphan_AMFCompType_CSTypeTransform {
from
s: MagicCRProfile!MagicCrCsiTemplate(
--Fire only if properEtfCT has a Component Types which is the child of
the transformed SUType of the aggregating SITemplate
s.properEtfCt->exists(ct|s.    magicCrBelongsToSiTemplate.
properEtfSUT->at(1).magicEtfGroups->includes(ct))
to
    t1: MagicAMFProfile!MagicSaAmfCtCSType(
        --Link to CompType
magicAmfSupportedby <- Set{ AMFCompTypeTransform(s.properEtfCt->
select(ct|s.    magicCrBelongsToSiTemplate. properEtfSUT->
at(1).magicEtfGroups->includes(ct))  ->at(1)},

```

```

        --Link to CStype
        magicSafSupportedCsType <- Set{ AMFCSTypeTransform(s.
magicCrCsiTempCSType)},
        --Transforming the attributes

    ),

t2 : MagicAMFProfile!MagicSaAmfSvcTypeCSType(
    --Link to SvcType
    magicAmfMemberOf <- Set{ s.magicCrBelongsToSiTemplate.
magicCrSiTempSvcType},
    --Link to CStype
    magicSafMemberCSType <- Set{ AMFCSTypeTransform(s.
magicCrCsiTempCSType)},
    --Transforming the Attributes
),

t3 : MagicAMFProfile!MagicSaAmfSutCompType(
    --Link to SUType
magicAmfMemberOf <- Set{s. magicCrBelongsToSiTemplate. properAmfSUT-
->at(1)},
    --Link to Component Type
magicSafMemberCompType <- Set{ AMFCompTypeTransform(s.properEtfCt
->select(ct|s.    magicCrBelongsToSiTemplate. properEtfSUT
->at(1).magicEtfGroups->includes(ct))) ->at(1)},

    t4: MagicCRProfile!MagicCrCsiTemplate(
        --Link to AMF CompType
properAmfCt <- Set{ AMFCompTypeTransform(s.properEtfCt->select(ct|s.
magicCrBelongsToSiTemplate. properEtfSUT->at(1).magicEtfGroups-
>includes(ct)) ->at(1)},
        --Link to CStype
        magicCrCsiTempCsType<- Set{ AMFCSTypeTransform(s.
magicCrCsiTempCSType)}
    )
}

```

Similarly *Orphan_AMFCompType_CSTypeTransform* refines a given CSITemplate and targets the generation of AMF CTypes and Component Types and their relationship. This rule fires only if all selected Component Types of the CSITemplate are orphan. Therefore, it simply transforms the first ETF Component Type selected for the current CSITemplate and the corresponding CStype to AMF Component Type and AMF CStype respectively. This transformation is performed by calling the *AMFCompTypeTransform*. *AMFCompTypeTransform* targets the generation of AMF Component Types and the attributes based on the corresponding ETF types. Moreover, *AMFCSTypeTransform*

generates the CStype associated with the current CSITemplate and initializes its attributes with the attributed specified by the corresponding ETF type.

```

rule Orphan_AMFCompType_CStypeTransform {
from
  s: MagicCRProfile!MagicCrCsiTemplate(
  --Fire only if all Component Types of properEtfCt are orphan
  not s.properEtfCt->exists(ct|s.    magicCrBelongsToSiTemplate.
properEtfSUT
  ->at(1).magicEtfGroups->includes(ct))

to
  t1: MagicAMFProfile!MagicSaAmfCtCStype(
    --Link to CompType
magicAmfSupportedby <- Set{ AMFCompTypeTransform(s.properEtfCt-
>at(1))},
    --Link to CStype
magicSafSupportedCsType <- Set{ AMFCStypeTransform(s.
magicCrCsiTempCStype)}
    --Transforming the attributes

  ),

  t2 : MagicAMFProfile!MagicSaAmfSvcTypeCStype(
    --Link to SvcType
magicAmfMemberOf <- Set{ s.magicCrBelongsToSiTemplate.
magicCrSiTempSvcType},
    --Link to CStype
magicSafMemberCStype <- Set{ AMFCStypeTransform(s.
magicCrCsiTempCStype)}
    --Transforming the Attributes
  ),

  t3 : MagicAMFProfile!MagicSaAmfSutCompType(
    --Link to SUType
magicAmfMemberOf <-Set{s.magicCrBelongsToSiTemplate.properAmfSUT-
>at(1)},
    --Link to Component Type
magicSafMemberCompType <- Set{AMFCompTypeTransform(s.properEtfCt-
>at(1))}
    --Transforming the Attributes
  ),

  t4: MagicCRProfile!MagicCrCsiTemplate(
    --Link to AMF CompType
properAmfCt <- Set{ AMFCompTypeTransform(s.properEtfCt->at(1))},
    --Link to CStype
magicCrCsiTempCsType<- Set{ AMFCStypeTransform(s.
magicCrCsiTempCStype)}
  )
do{

```

```

foreach(sut in s. magicCrBelongsToSiTemplate.
magicCrBelongsToSgTemplate. magicCrGroupsSiTemplates-
>collect(SITemp|SITemp.properAmfSUT->at(1)) {
    --Create the first end of the Link to SUType

    sut. magicSafMemberCompType <- sut. magicSafMemberCompType
->Union(Set{ AmfSutCompTypeCreate() })
    --Create the second end of the Link to Component Type
}
}

```

Similar to the previous rule, *t1*, *t2*, and *t3* create the required association classes, namely MagicSaAmfCtCSType, MagicSaAmfSvcTypeCSType, and MagicSaAmfSutCompType, between the newly generated AMF Types and the related created elements from the previous step. Finally, updating the CSITemplate with the list of AMF Component Types is performed by *t4*. Moreover, the *do* part of the transformation creates the link between created AMF component type and the previously generated AMF SUTypes of the sibling SITemplates of current CSITemplates SITemplate.

6.4 AMF Entity Creation

As shown in Figure 6-3, this phase takes as input the refined Configuration Requirements and the AMF model consisting of the generated AMF entity types. As a consequence of the previous transformation step, these models are connected by means of links defined among the AMF entity types (on one side) and the CSITemplates, SITemplates and SGTemplates (on the other side).

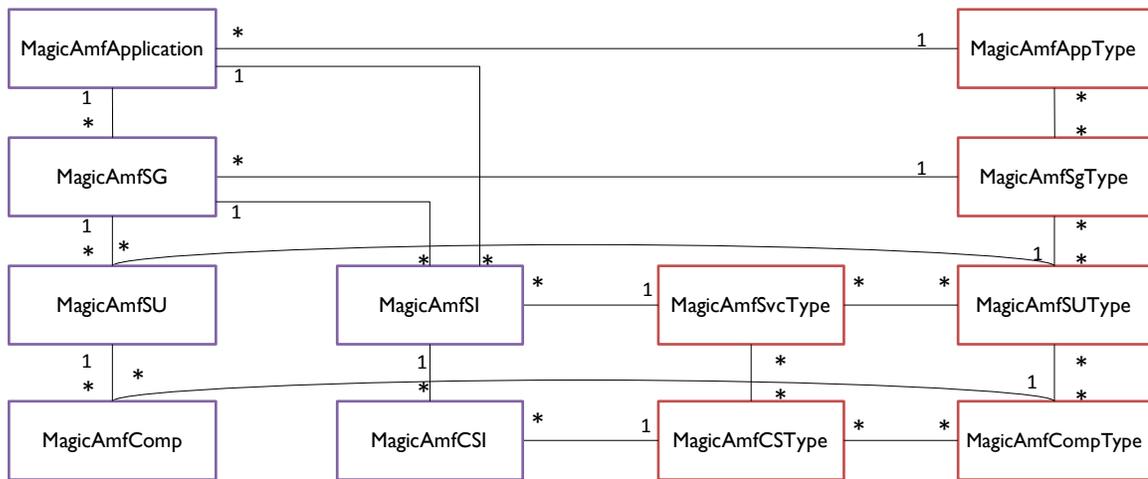


Figure 6-12 The result of the AMF Entity creation from the metamodel perspective

Similar to the generation of the entity types, the creation of entities starts from the Configuration Requirements elements. The generation of all the entities is driven by the characteristics of the entities types that have been created during the previous phase. The links defined between the configuration requirements elements and the AMF entity types ease the navigation of the AMF model favouring the direct access to most of the desired properties of such types. Figure 6-12 illustrates the result of this phase from the metamodel perspective. The generation follows an approach composed of three different steps. The first step targets the creation of different AMF entities, based on the entity types created in the previous phase, as well as establishing the relations among them. The second step aims at creating deployment entities. The third step prunes out all the Configuration Requirements elements as well as their links to the AMF configuration elements.

The result of this phase is a set of AMF entities and entity types which form an AMF configuration that satisfies the configuration requirements. In the following subsections we describe more in depth each transformation step.

6.4.1 Step 1: AMF Entity Instantiation

The main issue of this step consists of determining the number of entities that need to be generated for each identified entity type, and in defining the required links. For some entities we fetch this number directly from the Configuration Requirements model and for the others we need to calculate this number. In both cases the number of entities that need to be created depends on the values of the attributes specified in Configuration Requirement and AMF entity type elements.

Figure 6-13 shows the activity diagram which describes the flows of transformations performed in the context of this generation step. In the rest of this section we thoroughly describe these transformations.

This step starts with analyzing the SGTemplate and the AppType and SGType linked to the template and creates instances of entities compliant with the characteristics of these AMF types. It also generates the SUs providing the SIs that are protected by the generated SGs. Afterwards, the generation targets the definition of links between the generated entities, between the entities and the related types, and the generation of links between the SGTemplate and the generated entities.

The step is described by means of the following ATL code which consists of transformation rules and helper functions.

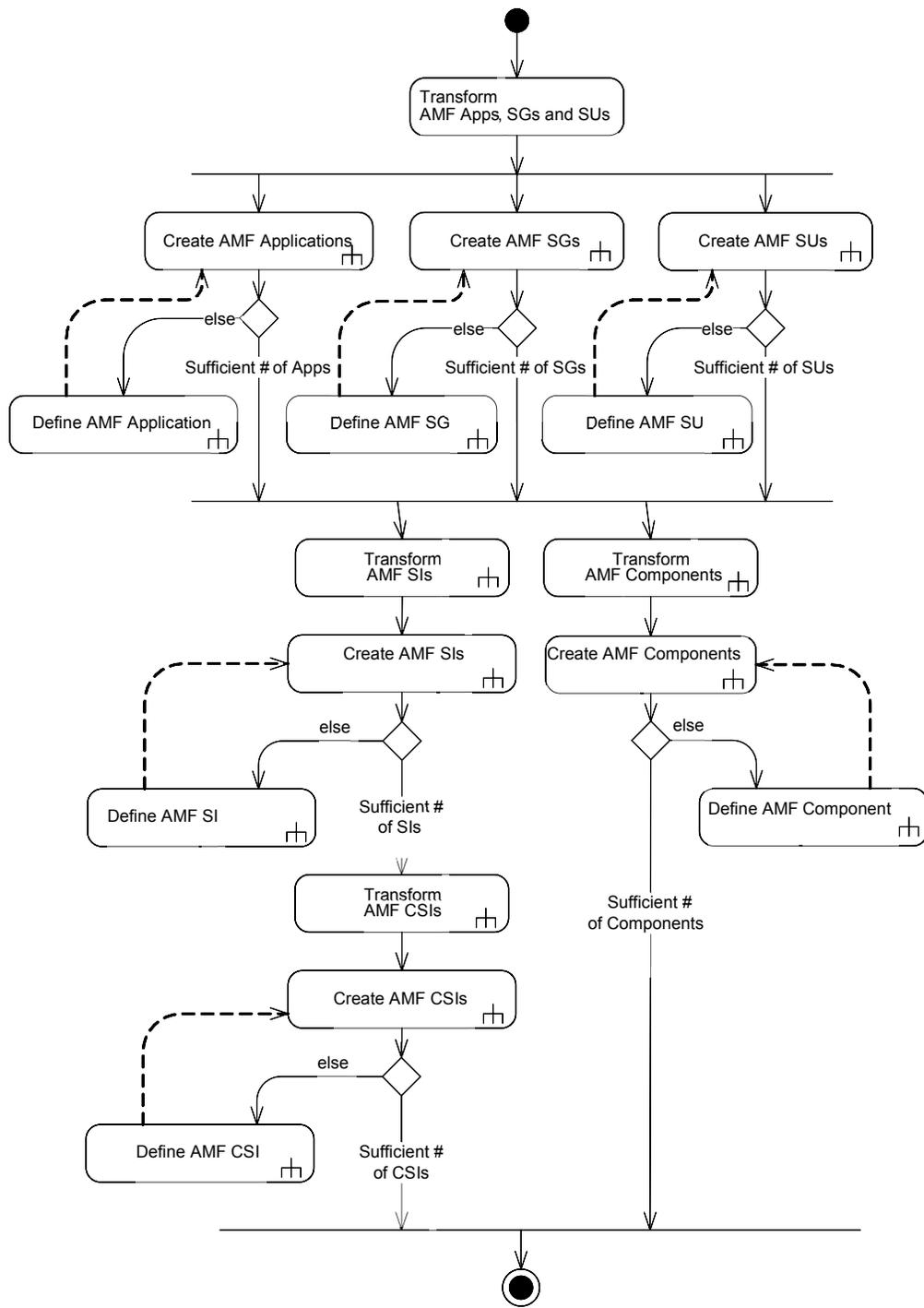


Figure 6-13 The flow of transformations to generate AMF entities

```

rule AMF_APP_SG_SU_Transform {
from
  s: MagicCRProfile! MagicCrSgTemplate

using{

  --Calculates the number of SGs

  maxNumSGs : Integer =
  s.magicCrGroupsSiTemplates
  ->iterate(sit, min:Integer = 0|
if sit.magicCrRegSiTempNumberofSis/sit.magicCrRegSiTempMinSis > min
then
  min= sit.magicCrRegSiTempNumberofSis/sit.magicCrRegSiTempMinSis
endif);

  --Calculates the number of SUs

  NumSUs : Integer = s.magicCRSgTempNumberofActiveSus+
  s.magicCRSgTempNumberofStdbSus+s.magicCRSgTempNumberofSpareSus;

  -- Calculates the total number of SIs
  TotalNumOfSIs : Integer =
    s. magicCrGroupsSiTemplates ->iterate(sitemp; num:Integer = 0| num
  + sitemp.magicCRSiTempNumberofSis);

  counter : Integer = 0;

  }

to
  t: MagicCRProfile! MagicCrSgTemplate (
    properAmfApp <-createAMFApplication(Set{},1),
    properAmfSG <- createAMFSG(Set{},NumOfSG),

  )

do {
  -- Create an Application and establish the link to SGs
  t.properAmfApp->at(1).magicAmfApplicationGroups <- t.properAmfSG;

  -- Establish the link from each SG to the aggregated SUs while
  creating them
  for (sg in t.properAmfSG){
    sg.magicAmfSGGroups <-
      createAMFSU(Set{}, NumSUs );
    t.properAmfSU <- t.properAmfSU->union(sg.magicAmfSGGroups);
  }
  -- Establish the link from each SU to the aggregated Components
  while creating them
  for (su in t.properSU){
    su.magicAmfLocalServiceUnitGroups <-
      s. magicCrSiTempGroups -> collect
    (e|AMF_Comp_Transform(e).properComp)
  }
}

```

```

}
-- Create the all required SIs
t.properAMFSI <-s. magicCrGroupsSiTemplates ->
collect(e|AMF_SI_Transform(e).properAmfSI);

-- Establish the link from each SG to the aggregated set of
protected SIs
for (sg in t.properAmfSG){
  sg.magicAmfSGProtects <- t.properAmfSI->asSequence()-
>subSequence(counter*TotalNumOfSIs/
maxNumSGs, (counter+1)*TotalNumOfSIs/ maxNumSGs);
  counter = counter +1;
}
}
}

```

The *AMF_APP_SG_SU_Transform* rule refines the *SGTemplate* by adding the links to the AMF entities namely Application, SG, and SU. These AMF entities are instantiated using different helper functions which take the required number of instances as an input and return the collection of AMF entities. For the Application there is only one instance needed for each *SGTemplate*, while for the case of SGs and SUs the number is calculated from the information specified in the *SGTemplate*. For instance, the definition of AMF Application uses the *createAMFApplication* helper function and a lazy rule called *APP_Define*. The helper function creates a set of AMF application entities in a recursive manner and in each recursion it calls the *APP_Define* lazy rule. *APP_Define* instantiates an AMF application entity, initializes its attributes starting from a given AMF AppType, and finally connects the generated entity to the type. Afterwards, the instantiated AMF Application is added to the set of entities and returns to the caller rule. The number of recursions corresponds to the number of required AMF applications specified by *AMF_APP_SG_SU_Transform* as an input. The same approach based on defining a helper function and a lazy rule is applied to create SGs and SUs.

The number of entities to be defined depends on the information which is specified in the Configuration Requirements model elements.

Once the proper entities are generated they are linked to the appropriate configuration entities. For instance, the generated SUs are grouped into different SGs depending on their capability of providing the SIs of a given type.

```
helper context MagicCRProfile! MagicCrSgTemplate
def: createAMFApplication (s:
Set (MagicAMFProfile!MagicSaAmfApplication), i: Integer) :
Set (MagicAMFProfile!MagicSaAmfApplication)=
  if i>0
  then
    let app: MagicAMFProfile!MagicSaAmfApplication =
      APP_Define (self.properSGT->at (1) .magicSaAmfSgtMemberOf->at (1))
    in
      self.createAMFApplication (s->union (app), i-1)
  else s
endif;

lazy rule APP_Define{
from
  s:MagicMagicProfile!MagicSaAmfAppType
to
  t:MagicMagicProfile!MagicSaAmfApplication(
    magicSafApp = CreateName(),
    magicSaAmfAppType <- s)
}
```

AMF_APP_SG_SU_Transform creates the link between newly generated AMF entities and connects them to the SGTemplate. Moreover, it creates the relation between the generated SGs and the protected set of SIs by means of the lazy rule *AMF_SI_Transform*. The rule is responsible for generating the required set of AMF SIs based on a given SITemplate. More specifically, *AMF_APP_SG_SU_Transform* uses *AMF_SI_Transform* for generating the SIs required by all the SITemplates aggregated by the SGTemplate. Using the same process, *AMF_APP_SG_SU_Transform* uses *AMF_Comp_Transform*

to generate the required components of each newly created SU and to connect them to the SU.

```

lazy rule AMF_Comp_Transform {
from
    s: MagicCRProfile! MagicCrCsiTemplate
using{
    --Calculates the number of components
    NumOfComp : Integer =
max(
self.magicCrBelongsToSiTemplate.magicCrBelongsToSgTemplate.magicCrGroup
sSiTemplates
->collect(sitemp|sitemp.magicCrSiTempGroups)
->select(csitemp|csitemp.magicCrCsiTempCsType =
self.magicCrCsiTempCsType)
->iterate(v, active:Integer = 0| active +
v.magicCrCsiTempNumberOfCsis*v.magicCrBelongsToSiTemplate.activeLoadper
SU)/
    (MagicSaAmfCtCsType.allInstances
    ->select(ctcst|ctcst.magicAmfSupportedby = s.properCt->at(1)
and ctcst.magicSafSupportedCsType =s.csiCSType)
    ->at(1). magicEtfMaxNumActiveCsi ,

    ->collect(sitemp|sitemp.magicCrSiTempGroups)
->select(csitemp|csitemp.magicCrCsiTempCsType =
self.magicCrCsiTempCsType)
->iterate(v, standby:Integer = 0| standby +
v.magicCrCsiTempNumberOfCsis*v.magicCrBelongsToSiTemplate.stdbLoadperSU
)/
    (MagicSaAmfCtCsType.allInstances
    ->select(ctcst|ctcst.magicAmfSupportedby = s.properCt->at(1)
and ctcst.magicSafSupportedCsType =s.csiCSType)
    ->at(1). magicEtfMaxNumStandbyCsi)-
    --Deducting the number of Component of the same Component Type created
for the other CSITemplates
self.magicCrBelongsToSiTemplate.magicCrBelongsToSgTemplate.magicCrGroup
sSiTemplates
->collect(sitemp|sitemp.magicCrSiTempGroups)
->select(csitemp|csitemp.properAmfCt->at(1) = self.properAmfCt->at(1))
->iterate(v, compNum:Integer = 0| compNum + v.properAmfComp->size())

);

}
to
    t: MagicCRProfile! MagicCrCsiTemplate (
        properAmfComp <- s.createAMFComp(Set{}, NumOfComp)
    )
}

```

The *AMF_Comp_Transform* rule refines the CSITemplate by adding the links from these templates to the AMF Components. The number of components that are required to

support the required number of CSIs is calculated based on the active load of the SU that will aggregate these components. This active load is calculated based on the required redundancy model expressed by the SGTemplate which contains the SITemplate that aggregate the current CSITemplate. The above presented code shows the part of the rule which calculates the number of components for the case of the N-Way redundancy model. The required set of AMF Components is generated by means of the helper function (*createAMFComp*) which takes the required number of components (*NumOfComp*) as an input. This helper function is similar to *createAMFApplication*.

After the generation of components, *AMF_APP_SG_SU_Transform* targets the definition of links between the required SIs and the newly created SGs which will protect them.

```

lazy rule AMF_SI_Transform {
from
    s: MagicCRProfile! MagicCrRegularSiTemplate

to
    t: MagicCRProfile! MagicCrRegularSiTemplate (
        properAmfSI <- s.createAMFSI(Set{},s.magicCRSiTempNumberofSis),
    )
do{
    for (si in t.properAmfSI){
        si.magicAmfSIGroups <- s.magicCrSiTempGroups
        -> collect (e|AMF_CSI_Transform(e).properAmfCSI);
If s.magicCrSiTempDependsOn->notEmpty()
then si.magicAmfDepends <- s.magicCrSiTempDependsOn->collect(sitemp|
sitemp.properAmfSI)
endif
    }
}
}

```

The generation of the SIs is described by the lazy rule *AMF_SI_Transform* shown above. This rule refines the SITemplate by adding the links to the set of SIs that is generated by means of the *createAMFSI* helper function. This function takes as input the required number of SIs and generates these entities using the same approach already described for

the case of AMF Application entities. *AMF_SI_Transform* invokes *createAMFSI* by passing the number of SIs which is specified by the SITemplate. Moreover, *AMF_SI_Transform* establishes the links between the newly generated SIs and their grouped CSIs by calling the *AMF_CSI_Transform* lazy rule. In addition, based on the dependency relationships specified in the SITemplate, this lazy rule establishes the dependency relationships between the newly created SIs and the SIs of the SITemplate on which the current SITemplate depends.

```

lazy rule AMF_CSI_Transform {
from
  s: MagicCRProfile!MagicCrCsiTemplate
to
  t: MagicCRProfile!MagicCrCsiTemplate(
    properAmfCSI <- s.createAMFCSI(Set{},s.magicCRCSiTempNumberOfCsIs)
  )
do{
  for (csi in t.properAmfCSI) {
    if s. magicCrcCsiTempDependsOn->notEmpty()
    then csi. magicSaAmfCSIDependencies<- s. magicCrCsiTempDependsOn
    ->collect(csitemp| csitemp. properAmfCSI)
    endif
  }
}
}

```

AMF_CSI_Transform (shown above) refines the CSITemplate by specifying the links between the template and the required set of CSIs. CSIs are generated invoking the *createAMFCSI* helper function which takes as input the required number of CSIs. *createAMFCSI* uses the same approach applied for the generation of AMF Application entities. *AMF_CSI_Transform* calls *createAMFCSI* by passing the number of CSIs expressed in the CSITemplate. Moreover, based on the dependency relationships specified in the CSITemplate, this lazy rule establishes the dependency relationships between the newly created CSIs and the CSIs of the CSITemplate on which the current CSITemplate depends.

6.4.2 Step 2: Generating Deployment Entities

After creating service provider and service entities based on the previously generated entity types, in this step we generate the deployment entities. Moreover, we deploy the service provider entities (e.g. SU) on deployment entities (e.g. Node). For the sake of simplicity, our approach assumes that all the nodes are identical and thus the SUs are distributed among nodes evenly. The number of nodes and their attributes are explicitly specified in the Configuration Requirements by means of the NodeTemplate element.

The creation of the deployment entities is supported by two different transformations that target the generation of AMF Nodes and AMF Cluster respectively. The following code shows an ATL implementation of these transformations rules.

```
rule AMF_Node_Transform {
  from
    s: MagicCRProfile! MagicCrNodeTemplate

  using{
    TotalNumOfSUs : Integer = MagicAmfLocalServiceUnit.allInstances()-
>size();
    counter : Integer = 0
  }

  to
    t: MagicCRProfile! MagicCrNodeTemplate (
      properAmfNode <-createAMFNode(Set{},s.magicCRNumberOfNodes),
      magicAmfBelongsTo <- AMF_Cluster_Transform(s.magicCRNodeBelongsTo)
    )
  do {
    for (node in t.properAmfNode){
      node.magicAmfConfigureFor <- MagicAmfLocalServiceUnit.allInstances()->
asSequence()->subSequence
(counter*TotalNumOfSUs/s. magicCRNumberOfNodes,
(counter+1)*TotalNumOfSUs/s. magicCRNumberOfNodes);
      counter = counter +1;
    }
  }
}

lazy unique rule AMF_Cluster_Transform {
  from
    s: MagicCRProfile!MagicCrCluster
```

```
to
  t: MagicCRProfile!MagicCrCluster(
    properAmfCluster <-createAMFCluster(Set{},1)
  )
}
```

Notice that similar to the above presented case, the generation uses the helper function to create the required number of AMF entities.

6.4.3 Step 3: Finalizing the Generated AMF Configuration

As previously presented in Figure 6-3, the result of this phase is a model which is an instance of the AMF sub-profile. Therefore, once all the required entities have been generated, the final step consists of removing all Configuration Requirements elements which were used to generate the AMF configuration. This step simply consists of copying (without any change) all the AMF configuration elements and the relationships among them while leaving out the Configuration Requirements elements. To this end, for each AMF configuration entity and entity type it is required to define a transformation rule. These rules simply move the attributes of each model element as well as the relationships among them to the target model (AMF configuration). These rules are rather straightforward and thus are not presented in this dissertation.

6.5 Limitations

In order to specify the requirements, we have used a new artefact, called configuration requirement models. However, the CR model specifies the services to be provided and protected as well as their properties using elements which are close to AMF configurations concepts. As such, in order to specify the configuration requirements, one needs to 1) have the knowledge of SA Forum specifications, and 2) specify the

requirements at a low level of abstraction close to AMF standard concepts and far from the usual user requirements. Our approach can be improved by designing a requirement engineering phase which processes the high level user requirements and refines them into configuration requirement elements.

6.6 Summary

In this chapter we discussed our model-based approach for the generation of AMF configurations. This approach is based on our modeling framework. The proposed approach overcomes the complexity of the generation process by raising the level of abstraction at which the configuration properties must be defined. Compared to the code-centric approach [Kanso 2008 and Kanso 2009], our model-based approach offers a simplified generation process with a reduction of potential errors or inconsistencies. More specifically, by using a model transformation technique and a declarative implementation style, these rules abstract from the operational steps that are necessary for generating target elements. These rules simply specify the characteristics of the elements that have to be created without imposing operational constraints on how the target elements need to be created.

We have designed our approach in a modular and stepwise manner in which each step is supported by a set of transformation rules. The input and output of each transformation is an instance of the sub-profiles. Therefore, the interfaces between the different generation steps are formally defined in terms of modeling artefacts. As a consequence, the proposed approach is flexible and can be easily extended and refined. We have also directly reused the domain knowledge that has been acquired and modeled in the three sub-profiles. Finally, we have implemented our model-driven configuration generation approach using

ATL, a widely known toolkit for model transformation. Using these de-facto standard technologies will certainly result in a higher wider acceptance of the approach.

In the next chapter we will illustrate the effectiveness of the model-based configuration generation through the design of an AMF configuration for an online banking system.

Chapter 7

Implementation of the Framework and Application

To demonstrate the effectiveness of AMF configuration management framework, we used our model-based framework to develop a configuration for an online banking system which allows customers to conduct financial transactions using a secure web interface. In this chapter, we first introduce our prototype tool. Then, we use it for the case study and start by presenting the description of the software entities in the domain of online banking through an instance of our ETF sub-profile. After, we present the description of the requirements of the system for which we aim to generate an AMF configuration. These requirements are captured as an instance of the CR sub-profile. Finally, we apply the model-based AMF configuration generation approach.

7.1 Implementation of the Model-based Framework

We implemented the AMF profile in the IBM Rational Software Architect (RSA) [IBM 2011]. RSA is a UML 2.0 based integrated software development environment which supports UML extension capabilities and is built on top of the Eclipse platform [Eclipse 2011a]. The combination of RSA and Eclipse Modeling Framework (EMF) [Eclipse 2011b] 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].

Finally, our choice of using RSA also lies in the conclusions of the study conducted by Amyot et al. [Amyot 2006]. The authors compared different UML 2.0 integrated software development environments which support the design of UML profiles. This comparison was based on the capabilities of the tools such as integration with other tools and the effort required for defining a profile. RSA was found to be one of the most complete tools in its category.

Our process for generating model-driven configuration was implemented using ATLAS Transformation Language (ATL). ATL [Jouault 2006], a model transformation language, constitutes part of the Atlas Model Management Architecture (AMMA) platform and was created in response to the OMG MOF2.0 /QVT RFP [OMG 2007c]. ATL is used in the transformation scheme shown in Figure 7-1, permitting the transformation of the source model M_s , an instance of the source metamodel MM_s , into the target model M_t , an instance of the target metamodel MM_t .

ATL is a hybrid language which supports both imperative and declarative programming styles. In addition to specifying the mappings between source and target model elements, ATL provides imperative constructs, which help in specifying the mappings that are not easily expressed in a declarative manner.

ATL is implemented as an Eclipse project and forms part of the Model-to-Model (M2M) Eclipse project [Eclipse 2011d], a sub-project of the Eclipse Modeling Project [Eclipse 2011c]. We have used the Eclipse ATL Integrated Development Environment (IDE), an Eclipse plug-in built on the top of EMF, to develop the model-based AMF configuration generation approach discussed in Chapter 6.

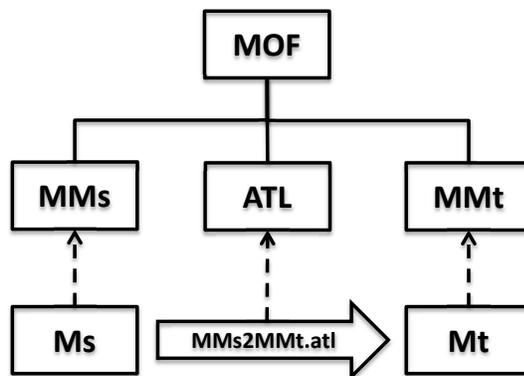


Figure 7-1 ATL Transformation scheme

7.2 The Online Banking System

Online banking is a system allowing users to perform banking activities via the internet. The features of this system include account transfers, balance inquiries, bill payments, and credit card applications. In this section we present the description of the software entities for online banking systems and, for this purpose, we have used our ETF sub-profile. It is worth noting that the ETF model for online banking system includes the description of the variety of software entities which can be used to design an online

banking application based on the requirements of the customer. This model often has different alternative software entities which can provide the same functionality. In fact, the AMF configuration generation is responsible for selecting the appropriate option which satisfies the configuration requirements.

7.2.1 The Billing Service

The electronic billing service is a feature of online banking 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. Figure 7-2 presents the ETF model for the billing system of our online banking software bundle. It consists of an SUType (Billing) which provides BillingService SvcType. "Billing" includes BillManager Component Type which provides services for viewing and paying bills (ViewBill and PayBill CTypes). ViewBill depends on the EPostCommunication Component Type and PayBill is sponsored by ExtenalAccountManager through its ExternalBankCommunication CType.

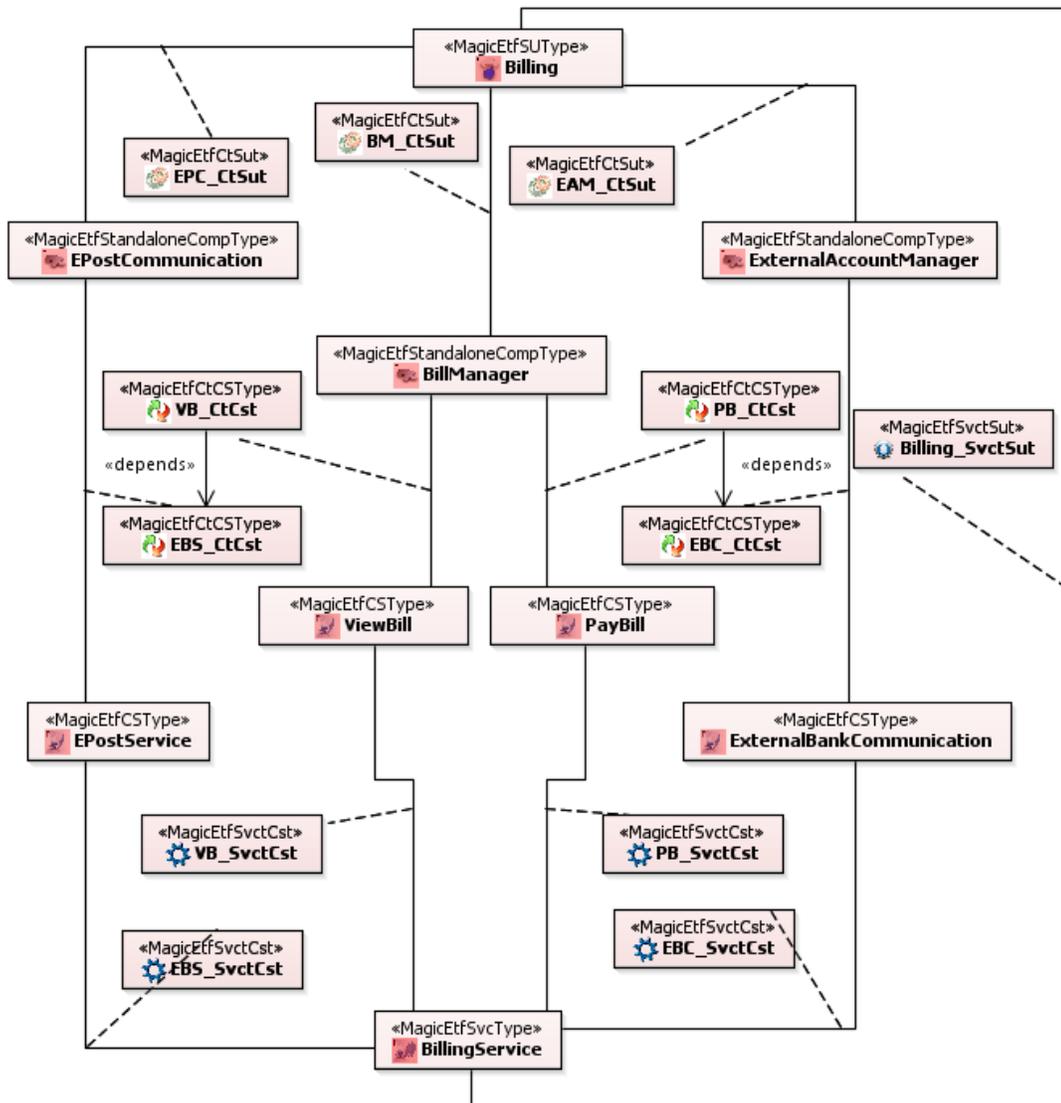


Figure 7-2 ETF model for billing part of an online banking software bundle

7.2.2 The Authentication Service

Security is one of the most important concerns for online banking systems. In our software bundle we have two different Component Types, namely CertifiedAuthentication and BasicAccessAuthentication, which provide the authentication service protecting clients' information (See Figure 7-3).

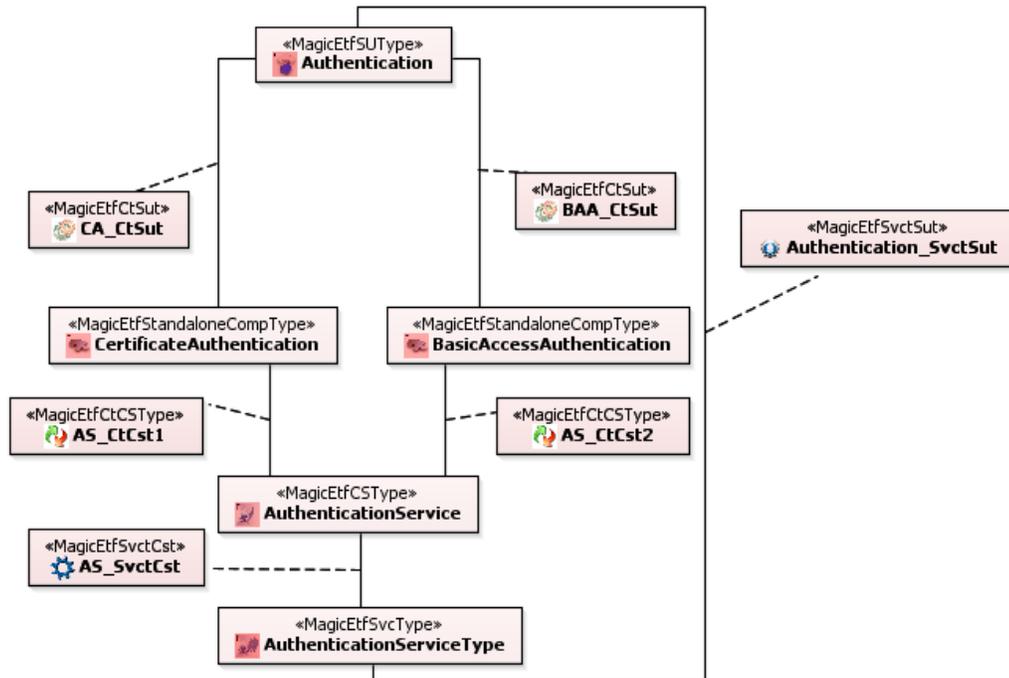


Figure 7-3 ETF model for the authentication part of an online banking software bundle

7.2.3 The Money Transfer Service

The fund transfer part of our sample online banking software bundle provides four different categories of money transfer services (see Figure 7-4):

- 1) Transferring money between the different accounts belonging to the same client (e.g. between saving and chequing accounts) which is provided by MoneyTransfer Component Type.
- 2) Performing money transfers from a client's account to another client's account(s) within the same banking institution. This service is provided by MoneyTransfer Component Type and is sponsored by the LocalAccountCommunication CSType of the ExternalAccountManager Component Type.

- 3) Performing money transfers from a client's account to an account held by a different banking institution. This service is provided by MoneyTransfer Component Type and is sponsored by ExternalAccountCommunication CStype of the ExternalAccountManager Component Type.
- 4) Transferring funds to the Visa account of a client which is supported by VisaPayment Component Type and is sponsored by VisaAccountCommunication CStype of the ExternalAccountManager Component Type.

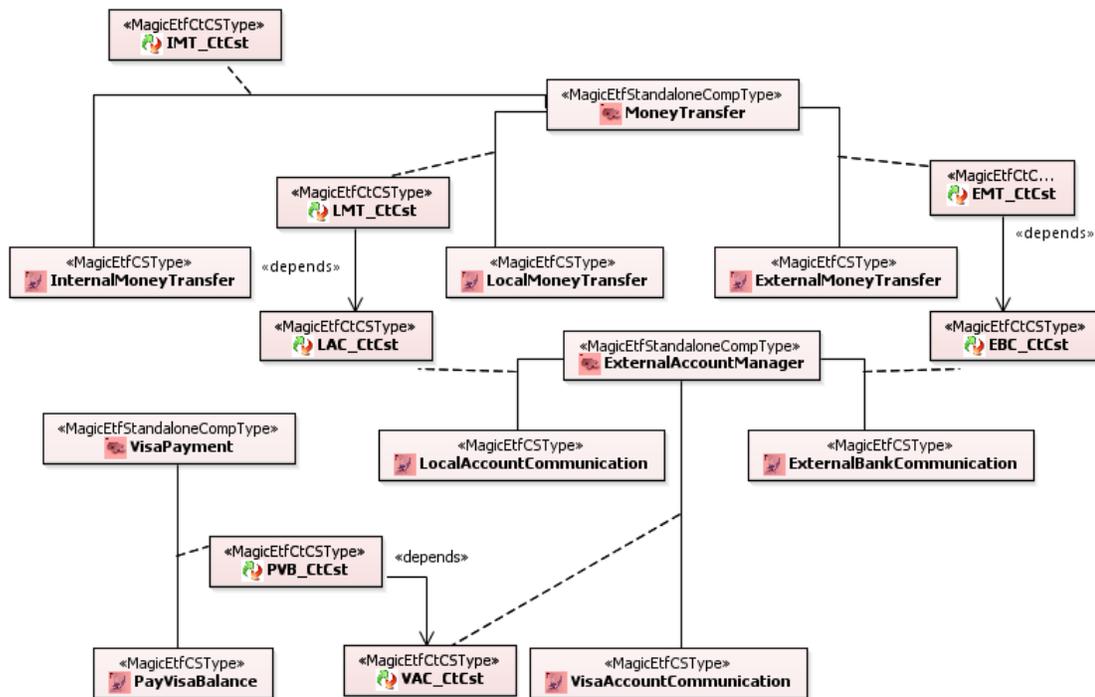


Figure 7-4 ETF model for money transfer part of online banking software bundle

7.2.4 Web Server and User Interface

In order to support the web based interface, the online banking software bundle includes two well-known solutions, Apache Web Server and IBM WebSphere, which are represented through two different ETF SUTypes in Figure 7-5. WebSphereServer

SUType includes WebSphere Component Type and ApacheServer groups Apache Component Type. Both Component Types provide the Web CStype which forms the WebServiceType SvcType. The difference between WebSphere and Apache Component Types lies in the component capability model for providing Web CStype. More specifically, the component capability model for Apache is MAGIC ETF COMP 1 ACTIVE while this attribute is equal to MAGIC ETF COMP X ACTIVE AND Y STANDBY for WebSphere. In other words, Apache has more limitations than WebSphere in providing the Web CStype (e.g. Apache cannot participate in an SU aggregated in an SG with N-Way redundancy model).

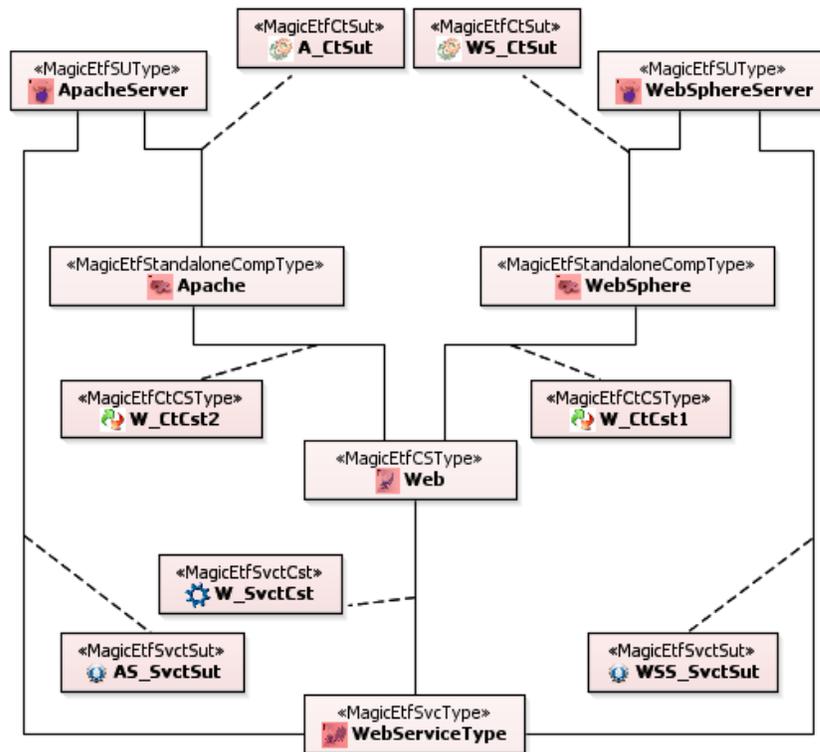


Figure 7-5 ETF model for web server part of online banking software bundle

The web based user interface of the online banking system consists of a set of web modules. In the ETF model in Figure 7-6 these web modules are presented in terms of ETF Component Types grouped into an SUType called UserInterface.

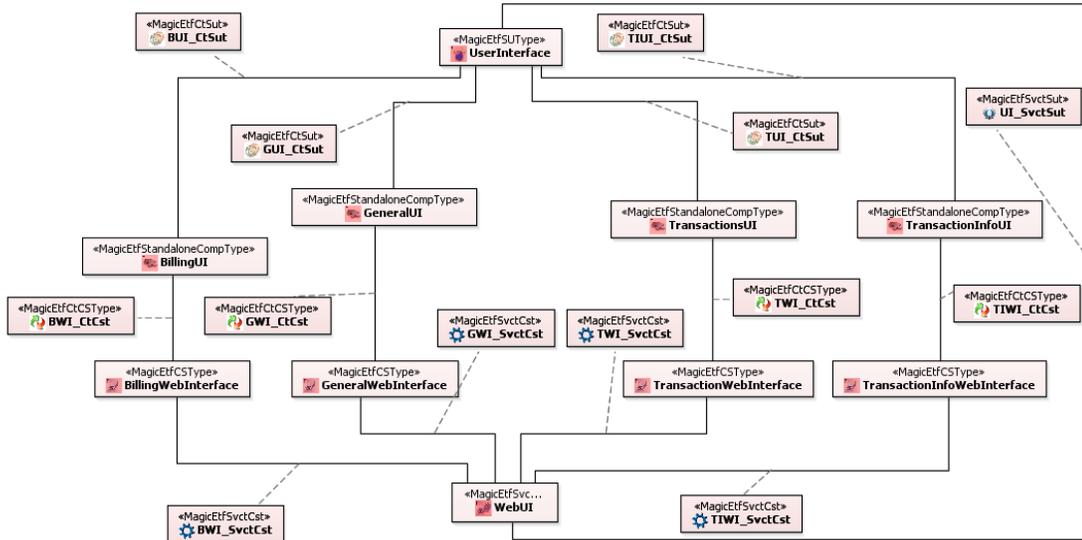


Figure 7-6 ETF model for user interface part of online banking software bundle

7.2.5 Database Management System

MySQL server and oracle server are included in the online banking software bundle and form the DBMS part of this bundle. They are both modeled in terms of ETF SUTypes (MySQLServer and OracleServer) and both provide the DataBaseManagement SvcType (See Figure 7-7).

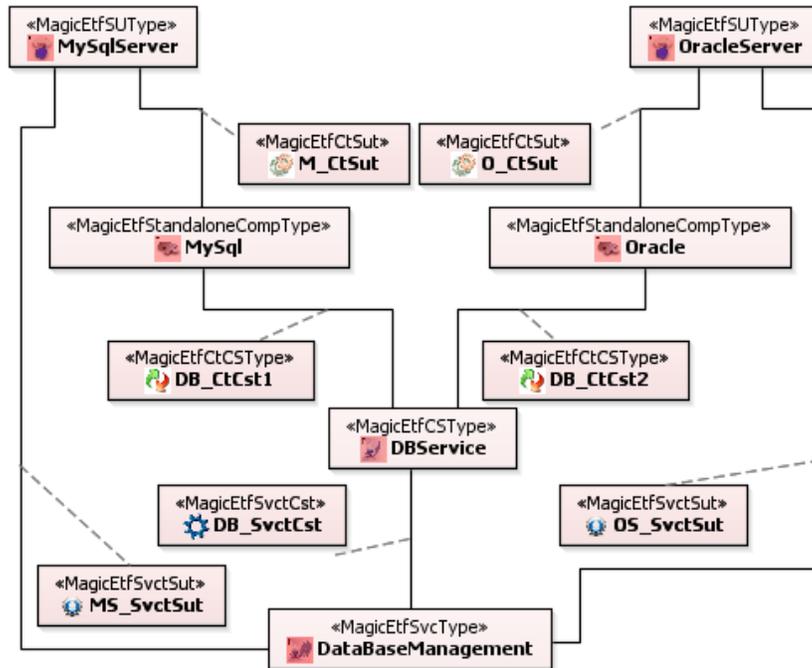


Figure 7-7 ETF model DBMS part of online banking software bundle

7.2.6 General Inquiries

The online banking software also includes a number of software entities providing services for public users such as financial advice, mortgage calculations, currency exchange information, and information about the various branches and ATM machines. In order to use these services, users do not need to be clients of the banking institution and, therefore, authentication is not necessary for them. Figure 7-8 represents the ETF model describing the software entities for general inquiries. Advice&Tools Component Type provides FinancialAdvice, MortgageCalculator, and CurrencyExchangeCalculator CTypes. General Information Provider Component Type provides the ATM/BranchLocator CType sponsored by the MapInformation CType which is provided by the GoogleMap Component Type.

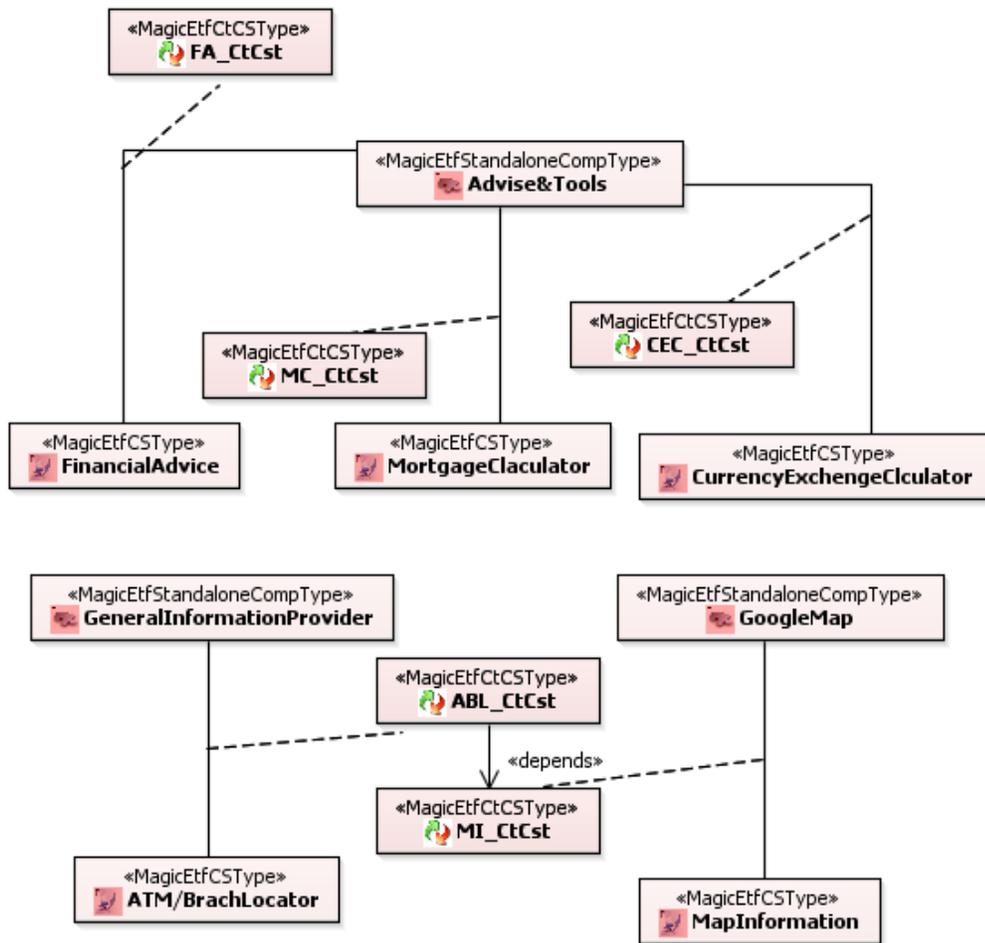


Figure 7-8 ETF model for the general inquiries part of an online banking software bundle

7.2.7 Transaction Information

One of the most useful services in online banking systems involves providing information concerning the recent transactions of the client's account. Some examples of such services include viewing recent transactions, downloading bank statements, and viewing images of paid cheques. The ETF elements of providing these services are presented in Figure 7-9.

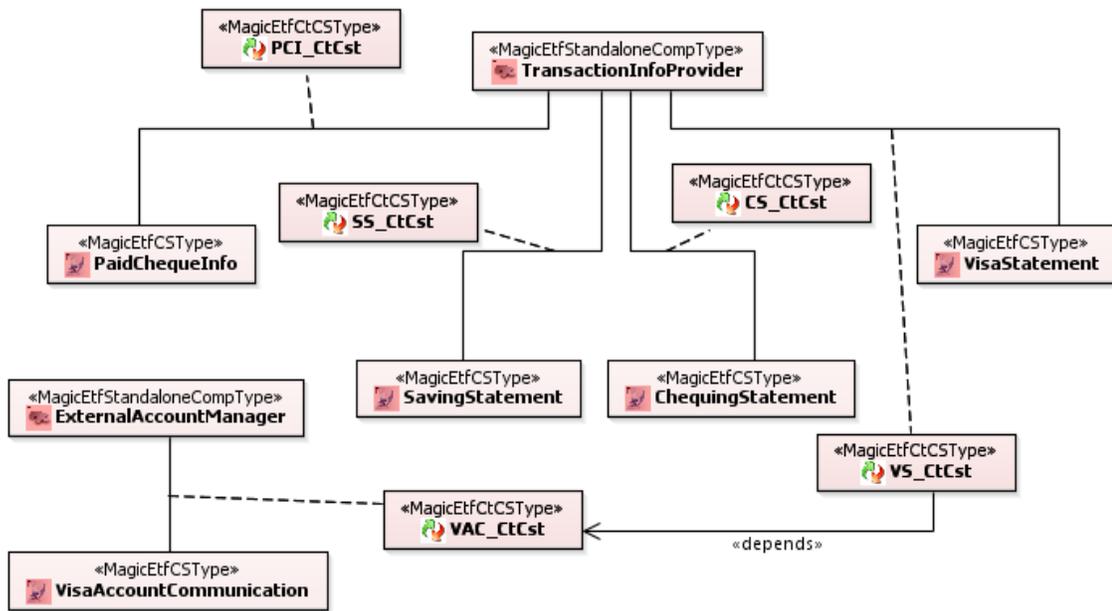


Figure 7-9 ETF model for the transaction information part of an online banking software bundle

7.2.8 SUType Level Dependency

The dependency between SUTypes of an online banking system is shown in Figure 7-10. In particular, providing UserInterface service WebUI SUType depends on the provision of the WebServiceType SvcType. The DataBaseManagement SvcType sponsors the provision of the AuthenticationServiceType by Authentication SUType.

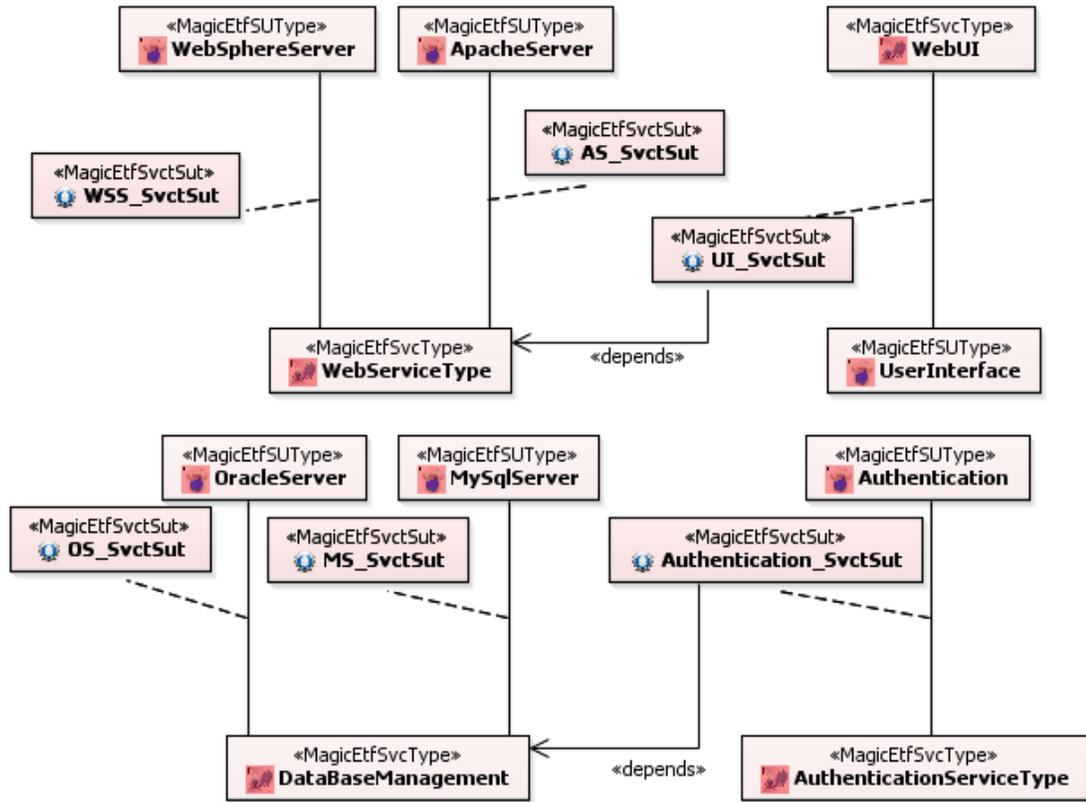


Figure 7-10 SUType level dependency

7.3 Configuration Requirements for the Online Banking System

The ETF model of the previous section describes the software which contains the software entities for online banking systems. It often includes different software components for providing the same services and thus includes different alternative solutions. For instance, the number of active/standby assignments that two different components can support for providing the same functionality may vary. This may make one software entity an appropriate match for satisfying configuration requirements over other possible alternatives.

The requirements needed to be satisfied by an AMF configuration of a given application are specified in a configuration requirement model, i.e. an instance of the CR sub-profile.

In this section we specify the configuration requirements of a specific imaginary online banking system called Safe Bank. The configuration requirements are defined based on the high level requirements specified by stakeholders of Safe Bank. In other words, it is the responsibility of the software analyst to extract configuration requirements from the software requirement specification. It is worth noting that the process of refining software requirements into configuration requirements is beyond the scope of this thesis. Therefore, in this section we only present the results of this refinement process i.e. the configuration requirement model. In the following sections, using our model-based configuration generation method and basing our approach on the software bundle presented in Section 7.2, we generate an AMF configuration for the Safe Bank online banking system which satisfies these requirements.

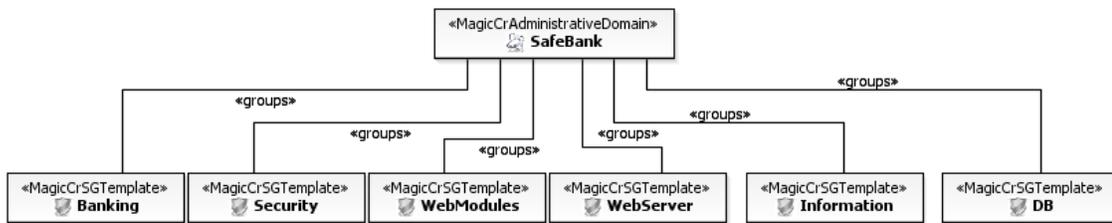


Figure 7-11 The SGTemplates of the Safe Bank online banking system

Figure 7-11 shows the SGTemplates of the configuration requirement model for this system grouped in an Administrative Domain element called Safe Bank. The values of the attributes for each SGTemplate are represented in Table 7-1. These attributes specify the requirements of the redundancy model for each SGTemplate and are extracted from software requirement specification. For instance, for more critical SGTemplates such as Security and DB, the required redundancy model is N-Way which supports a higher level

of service protection. On the contrary, the 2N redundancy model is specified for less critical SGTemplates, e.g. Webmodules and Information.

Table 7-1 List of values of attributes of the SGTemplates specified for the Safe Bank online banking system

Attribute \ SGTemplate	Information	Banking	Security	DB	WebServer	WebModules
magicCrSgTempRedundancyModel	2N	N+M	N-Way	N-Way	N+M	2N
magicCrSgTempNumberofActiveSus	1	2	3	3	3	1
magicCrSgTempNumberofStdbSus	1	1	0	0	1	1

WebModules defines the requirements for the SG responsible for protecting the services provided at the web user interface level. It consists of Private and Public SITemplates which depend on the WebServerService SITemplate of WebServer SGTemplates (see Figure 7-12). Table 7-2 presents the values of the attributes of these SITemplates and their aggregated CSITemplates. The Public SITemplate models the requirements of the UI services needed to be provided for system users who are not necessarily Safe Bank clients. The Private SITemplate, on the other hand, defines the requirements of the UI services provided only for Safe Bank clients. It consists of two CSITemplates, TransactionUI and TransactionInfoUI, which specify the configuration requirements of the user interface for transactional services and statement information services, respectively. Once again, the values of these attributes are specified as a result of the requirement refinement performed by the software analyst. For instance, the required number of active/standby assignments is defined based on the required level of protection. The number of SIs, however, is specified based on the expected workload in the system. Since the Public SITemplate specifies the part of the system which is visible for both authorized and unauthorized users, the number of SIs is twice the number of SIs

specified for the Private SITemplate which is only accessible for authorized users. Note that the value of the additional attributes (expectedSIsperSG, activeLoadperSU, and stdbLoadperSU) are calculated and populated using the *CR_Preprocessing* rule from the previous chapter and are based on the parameters specified in the CR model.

Table 7-2 List of the values of attributes of SITemplates and CSITemplates of WebModules and WebServer SGTemplates

Attribute \ SITemplate	Public	Private		WebServerService
	magicCrSiTempSvcType	WebUI	WebUI	
magicCrSiTempNumberOfActiveAssignments	1	1		1
magicCrSiTempNumberOfStdbAssignment	1	1		1
magicCrRegSiTempNumberOfSis	20	10		5
magicCrRegSiTempMinSis	10	10		5
<i>expectedSIsperSG(Calculated)</i>	<i>10</i>	<i>5</i>		<i>5</i>
<i>activeLoadperSU(Calculated)</i>	<i>10</i>	<i>5</i>		<i>2</i>
<i>stdbLoadperSU(Calculated)</i>	<i>10</i>	<i>10</i>		<i>5</i>
Attribute \ CSITemplate	GeneralUI	TransactionUI	TransactionInfoUI	Web
	magicCrCsiTempCsType	GeneralWebInfo	Transaction WebInterface	TransactionInfoWeb Interface
magicCrCsiTempNumberOfCsis	1	1	1	1

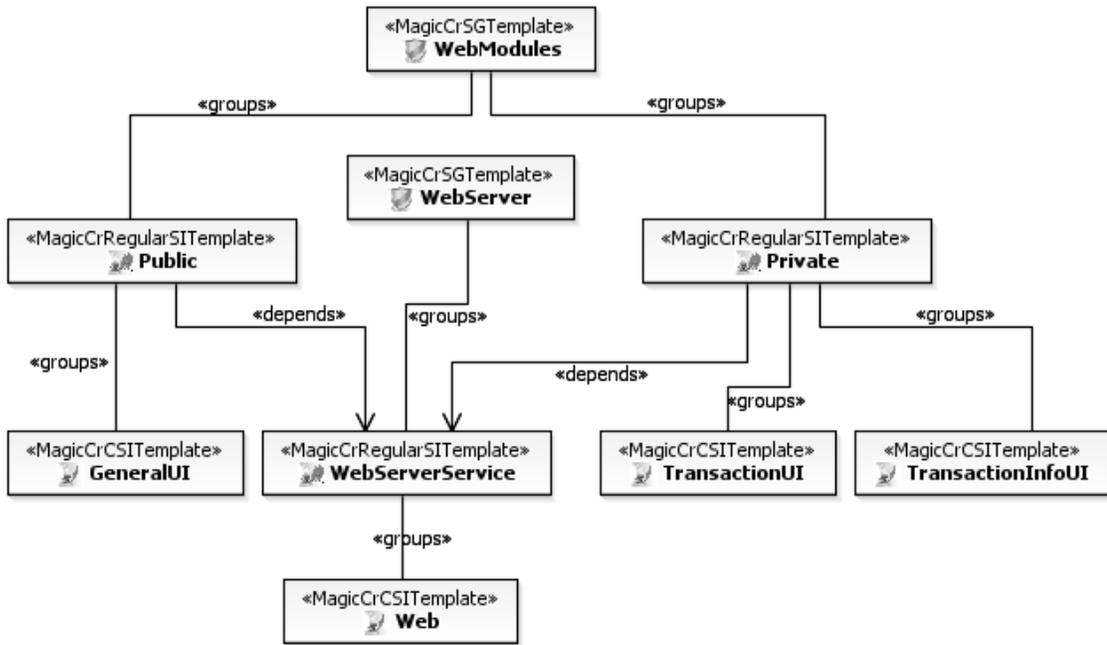


Figure 7-12 Configuration requirement elements of WebModules and WebServer SGTemplates

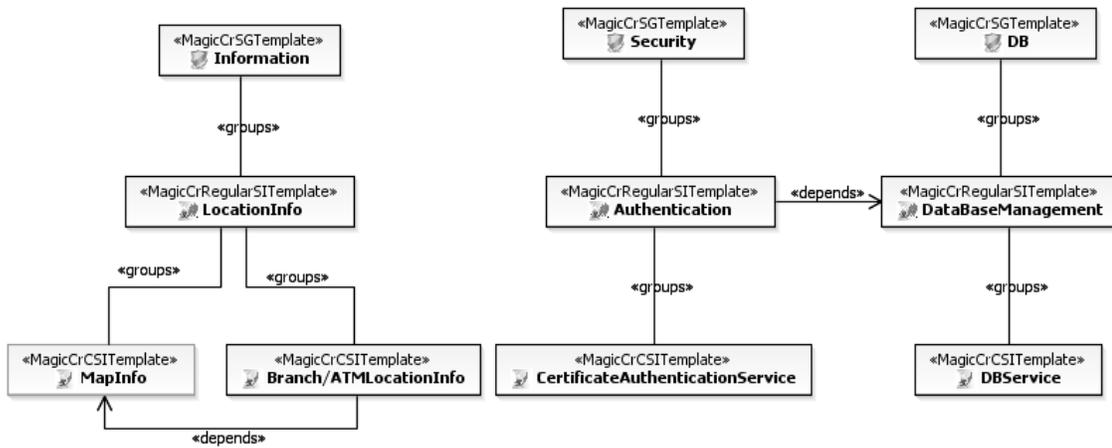


Figure 7-13 Configuration requirement elements of Security, Information, and DB SGTemplates

The configuration requirement elements defined for Security, Information, and DB SGTemplates are illustrated in Figure 7-13 and the values of their attributes are specified in Table 7-3.

Table 7-3 List of the values of attributes of SITemplates and CSITemplates of Security, Information, and DB SGTemplates

Attribute \ SITemplate	Authentication	LocationInfo		DatabaseManagement
magicCrSiTempSvcType	AuthenticationService Type	GeneralInquiries		DatabaseManagement
magicCrSiTempNumberOfActiveAssignments	2	1		2
magicCrSiTempNumberOfStdAssignment	1	1		1
magicCrRegSiTempNumberOfSis	5	1		5
magicCrRegSiTempMinSis	5	1		5
<i>expectedSIsperSG(Calculated)</i>	5	1		5
<i>activeLoadperSU(Calculated)</i>	5	1		5
<i>stdbLoadperSU(Calculated)</i>	3	1		3
Attribute \ CSITemplate	CertificateAuthentication Service	Branch/ATM LocationInfo	MapInfo	DBService
magicCrCsiTempCsType	AuthenticationService	ATM/ BranchLocator	MapInfo rmation	DBService
magicCrCsiTempNumberOfCsis	1	1	1	1

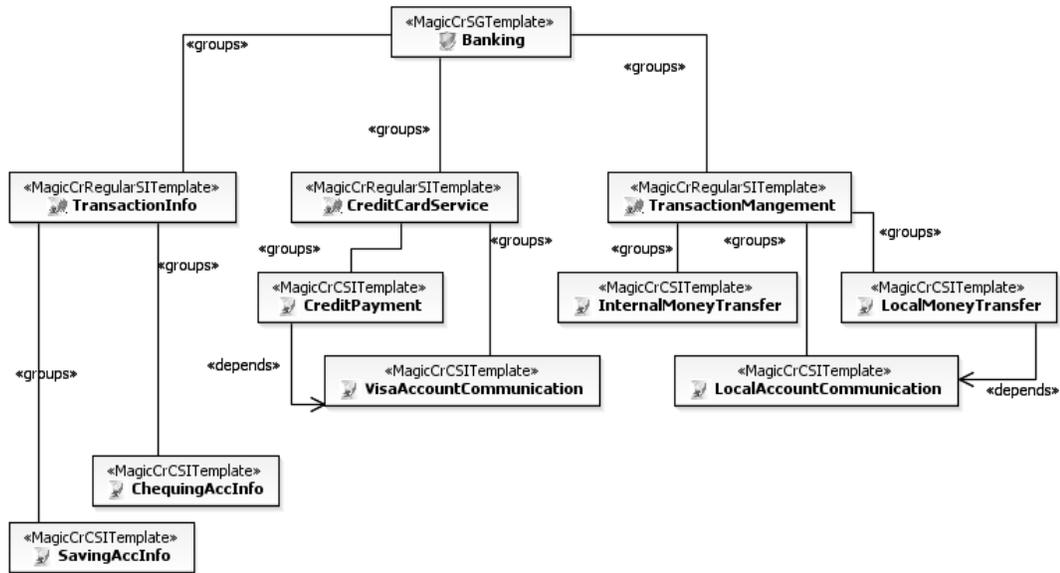


Figure 7-14 Configuration requirement elements of Banking SGTemplate

The configuration requirement elements defined for Banking SGTemplate are illustrated in Figure 7-14 and the values of their attributes are specified in Table 7-4. Banking SGTemplate specifies three different SITemplates:

- TransactionManagement, which specifies the configuration requirements for money transfer services, i.e. internal money transfers between a client’s accounts and local money transfers for transferring money between two different Safe Bank clients.
- CreditCardService, characterizing the required transactions of credit cards limited to credit card balance payments in the Safe Bank system.
- TransactionInfo, which models the requirements of different account information services.

Table 7-4 List of the values of attributes of SITemplates and CSITemplates of Banking SGTemplates

Attribute \ SITemplate	Transaction Management		
magicCrSiTempSvcType	TransactionService		
magicCrSiTempNumberOfActiveAssignments	1		
magicCrSiTempNumberOfStdbAssignment	1		
magicCrRegSiTempNumberOfSis	1		
magicCrRegSiTempMinSis	1		
<i>expectedSIsperSG(Calculated)</i>	<i>1</i>		
<i>activeLoadperSU(Calculated)</i>	<i>1</i>		
<i>stdbLoadperSU(Calculated)</i>	<i>1</i>		
Attribute \ CSITemplate	LocalMoneyTransfer	InternalMoneyTransfer	LocalAccountCommunication
magicCrCsiTempCsType	LocalMoneyTransfer	InternalMoneyTransfer	LocalAccountCommunication
magicCrCsiTempNumberOfCsis	1	1	1

Attribute \ SITemplate	CreditCard Service		TransactionInfo	
	magicCrSiTempSvcType	TransactionService		TransactionInfo
magicCrSiTempNumberOfActiveAssignments	1		1	
magicCrSiTempNumberOfStdbAssignment	1		1	
magicCrRegSiTempNumberOfSis	1		2	
magicCrRegSiTempMinSis	1		2	
<i>expectedSIsperSG(Calculated)</i>	1		2	
<i>activeLoadperSU(Calculated)</i>	1		1	
<i>stdbLoadperSU(Calculated)</i>	1		2	
Attribute \ CSITemplate	Credit Payment	VisaAccount Communication	Saving AccInfo	Chequing AccInfo
	magicCrCsiTempCsType	PayVisaBalance	VisaAccount Communication	Saving Statement
magicCrCsiTempNumberOfCsis	1	1	1	1

The required deployment infrastructure is specified in terms of NodeTemplate and the properties of the cluster are modeled using an element called Cluster. The configuration requirement for the deployment infrastructure consists of one Cluster and one NodeTemplate which implies that all nodes of the cluster are identical. The number of required nodes equals to 10 and Figure 7-15 shows the CR elements for deployment infrastructure.



Figure 7-15 Configuration requirements for deployment infrastructure

7.4 Generation of an AMF Configuration for Safe Bank Online Banking System

7.4.1 Selecting ETF Types

The selection of ETF types is performed based on the rules in the steps presented in Section 6.2 and considering the selection criteria: service provision, the component capability model, the redundancy model, the load of the SUs, and the dependency between different elements used to provide services. For instance, in the CR model, DBService CSITemplate specifies the required CSType as DBService and thus, both Oracle and MySql ETF Component Types can be selected for this CSITemplate (see dashed lines in Figure 7-16). The required service type specified through the parent SITemplate is DatabaseManagement which is also supported by OracleServer and MySqlServer SUTypes. However, the redundancy model specified by DB SGTemplate is N-Way, requiring that the Component Types have the component capability model of MAGIC_ETF_COMP_X_ACTIVE_AND_Y_STANDBY which is only supported by the Oracle Component Type. Therefore, the MySql Component Type is removed from the set of appropriate Component Types of the DBService CSITemplate.

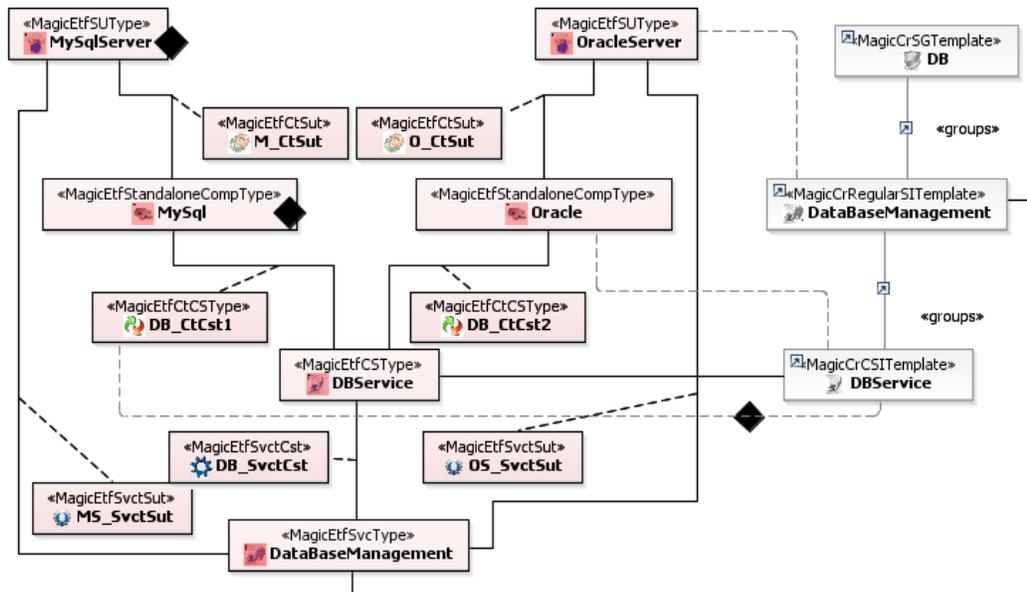


Figure 7-16 ETF Type selection phase for the DBMS part of online banking ETF

Since OracleServer provides the required SvcType and supports the required load, OracleServer SUType is selected for DatabaseManagement SITemplate in the SITemplate refinement step. Figure 7-16 shows the effect of the ETF Type Selection transformation step on the DBMS part of online banking ETF. Seeing as the elements marked by the black diamond do not satisfy all specified requirements, they will be pruned out of the model.

Figure 7-17 shows another example of applying the ETF Type Selection step by performing it on part of the Banking SGTemplate. In this figure the dashed lines connect the selected ETF type for each CR element. Since the MoneyTransfer part of our ETF model does not include any SUTypes, this phase only selects appropriate Component Types for CSITemplates. To this end, MoneyTransfer Component Type has been selected for both LocalMoneyTransfer and InternalMoneyTransfer CSITemplates due to the

provision of InternalMoneyTransfer and LocalMoneyTransfer CTypes by this Component Type. ExternalAccountManager Component Type has been selected for LocalAccountCommunication CSITemplate in order to provide the service necessary for managing the communication between the accounts of Safe Bank’s clients. It is worth noting that the dependency relationship between LocalMoneyTransfer and LocalAccountCommunication CSITemplates is compliant with the dependency between LMT_CtCst and LAC_CtCst ETF elements (see Figure 7-17). Therefore, the selected ETF types successfully pass refinement step based on SI dependency presented in Section 6.2.4.

Similarly, the ETF type selection phase is performed on the rest of the CR model elements, but will be omitted for the sake of avoiding repetition.

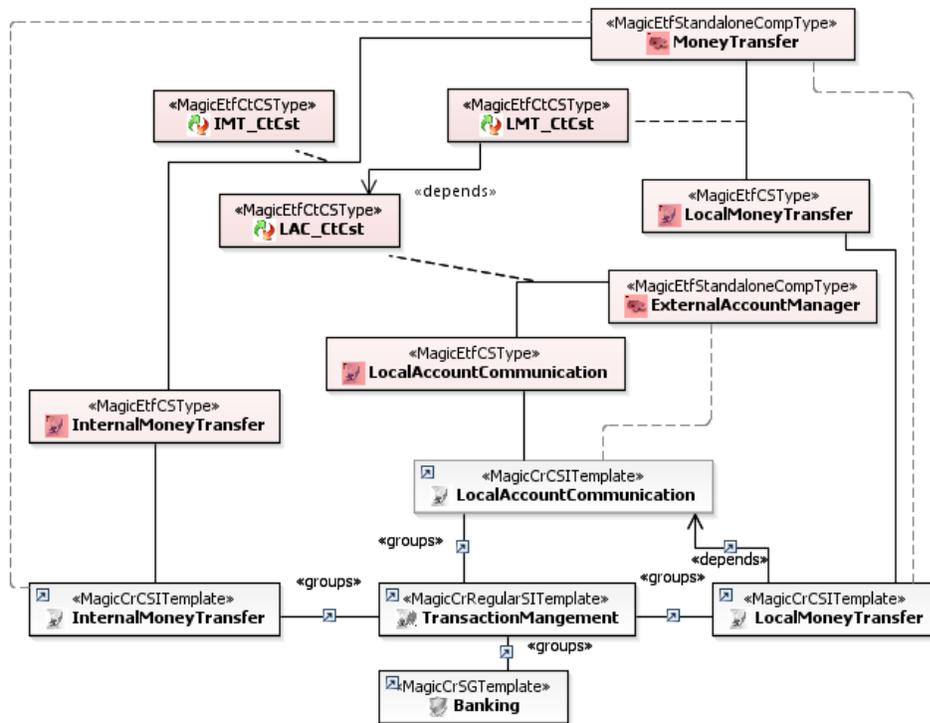


Figure 7-17 ETF Type selection phase for TransactionManagement SITemplate

Another example of the AMF type creation phase for TransactionManagement SITemplate is presented in Figure 7-19, Figure 7-20, and Figure 7-21. Figure 7-19 shows the creation of the Banking AMF SGType for the Banking SGTemplate as well as the generation of TransactionManagement AMF SUTypes and TransactionService AMF SvcType for TransactionManagement SITemplate. It is worth noting that, since the ETF model does not include any ETF SUTypes or any ETF SGTypes, the generation of the respective AMF types is performed from scratch.

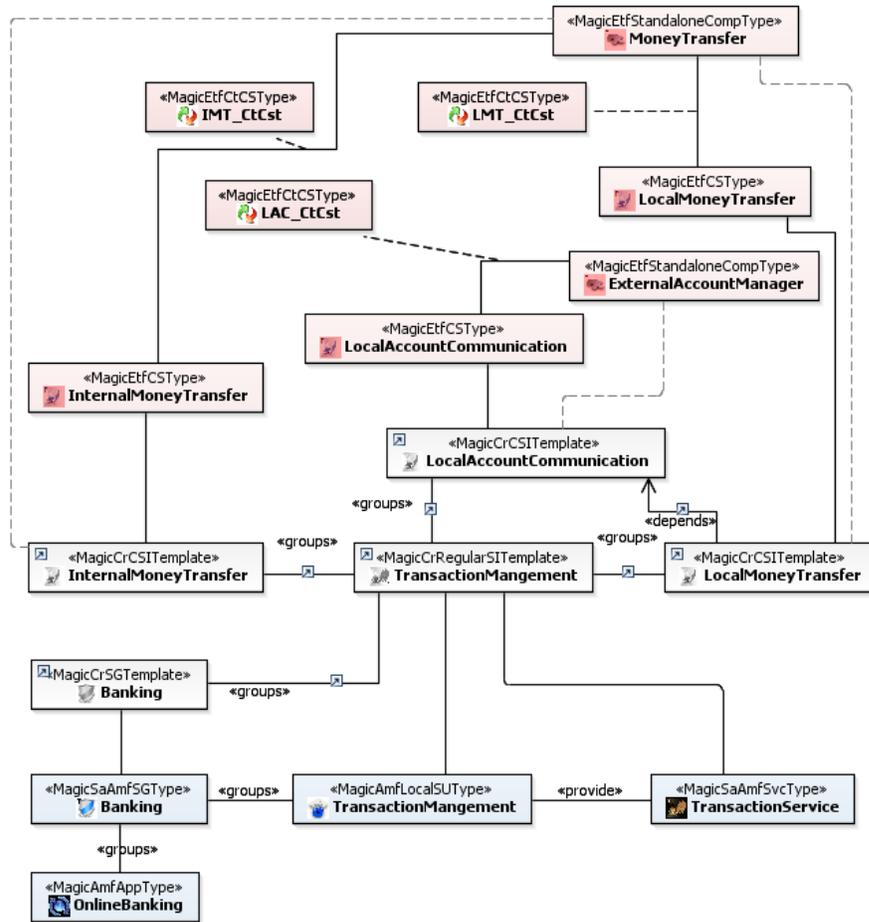


Figure 7-19 AMF SGType, AMF SUType, and AMF SvcType generation steps for TransactionManagement SITemplate and Banking SGTemplate

Figure 7-20 presents the result of the AMF Component Type and CStype generation phase (see Section 6.3.3) for the CSITemplates of the TransactionMangement SITemplate. In this step the AMF types are generated based on the selected ETF types which resulted from the ETF type selection phase. For purposes of clarity, in Figure 7-20 uses the same names for both ETF types and their respective generated AMF types. Finally, Figure 7-21 shows the generated AMF types and the relationships created between them for TransactionManagement SITemplate as well as its parent SGTemplate and its CSITemplates resulting from the AMF type creation phase.

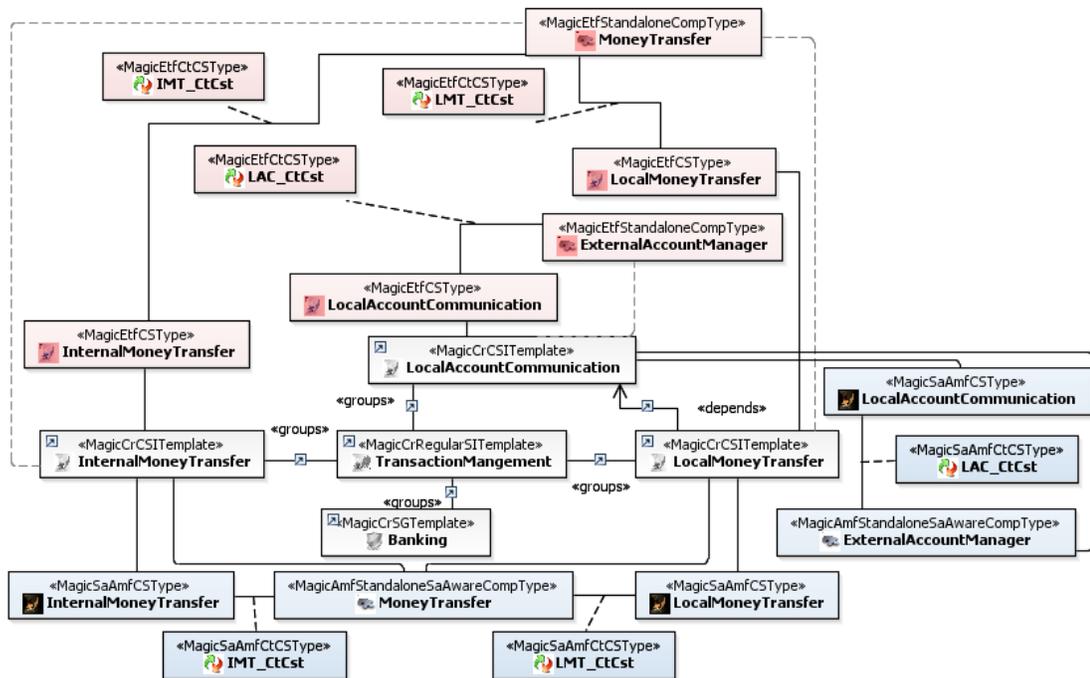


Figure 7-20 AMF Component Type and AMF CStype generation steps for the CSITemplates of TransactionManagement SITemplate

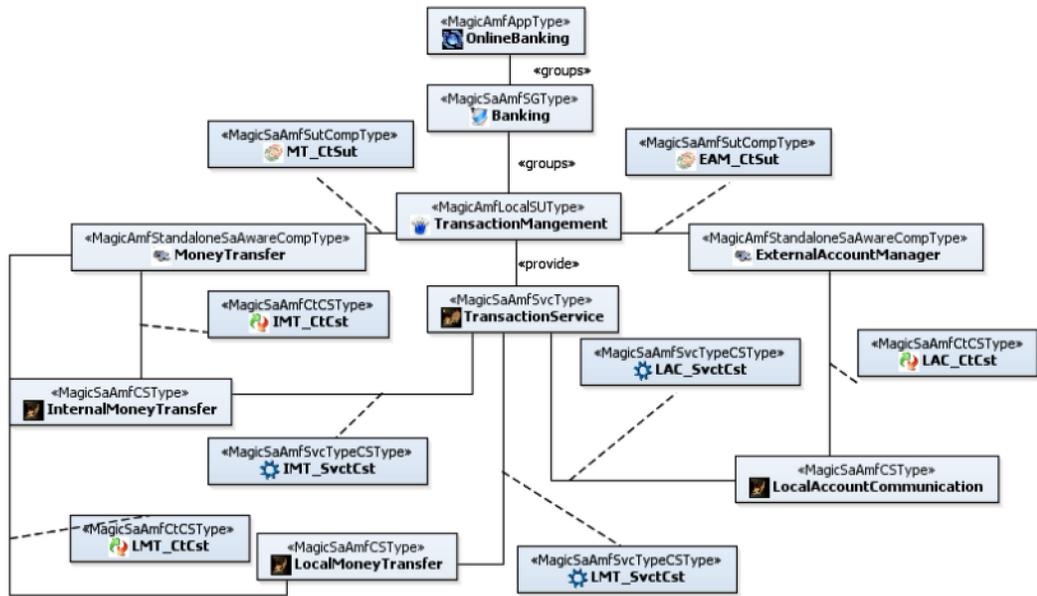


Figure 7-21 Created AMF Types for the transaction management part of online banking configuration

7.4.3 Creating AMF Entities

After creating the AMF entity types, the final phase of the transformation concerns creating the AMF entities for each previously defined AMF entity type based on the information captured by the Configuration Requirements. More specifically, the CR model specifies a set of requirements from which our model-based approach extracts the number of AMF entities necessary to be created. In Section 6.4.1, we specified the ATL rules for calculating the number of entities to be generated. In this section we present the required number of AMF entities for the part of the configuration concerning the DBMS service of Safe Bank’s online system. DB SGTemplate has only one SITemplate, DatabaseManagement, and in this SITemplate the minimum number of SIs and the number of required SIs are equal to 5. Therefore, the number of required SGs to be created is equal to one. As specified in DB SGTemplate (see Table 7-1), the required SG

should support the N-Way redundancy model and the number of member SUs equals 3. The number of components to be generated in each SU is calculated based on the capability of each component in providing CSIs in active and in standby mode. In the ETF model such a capability is described in the association class between Component Type and CStype (i.e. MagicEtfCtCStype) in terms of magicEtfMaxNumActiveCsi and magicEtfMaxNumStandbyCsi attributes. The value of these attributes is transformed into the attributes of its respective AMF type i.e. MagicSaAmfCtCStype. In this example the value of both attributes is equal to 3 and specified in the DB_CtCst association class between the Oracle AMF Component Type and DBService AMF CStype. To this end, based on the calculations specified in the ATL rules of Section 6.4.1, the number of components of each SU is equal to 2. The number of SIs and CSIs to be generated in the configuration are specified explicitly according to SITemplate and CSITemplate elements and can be easily extracted.

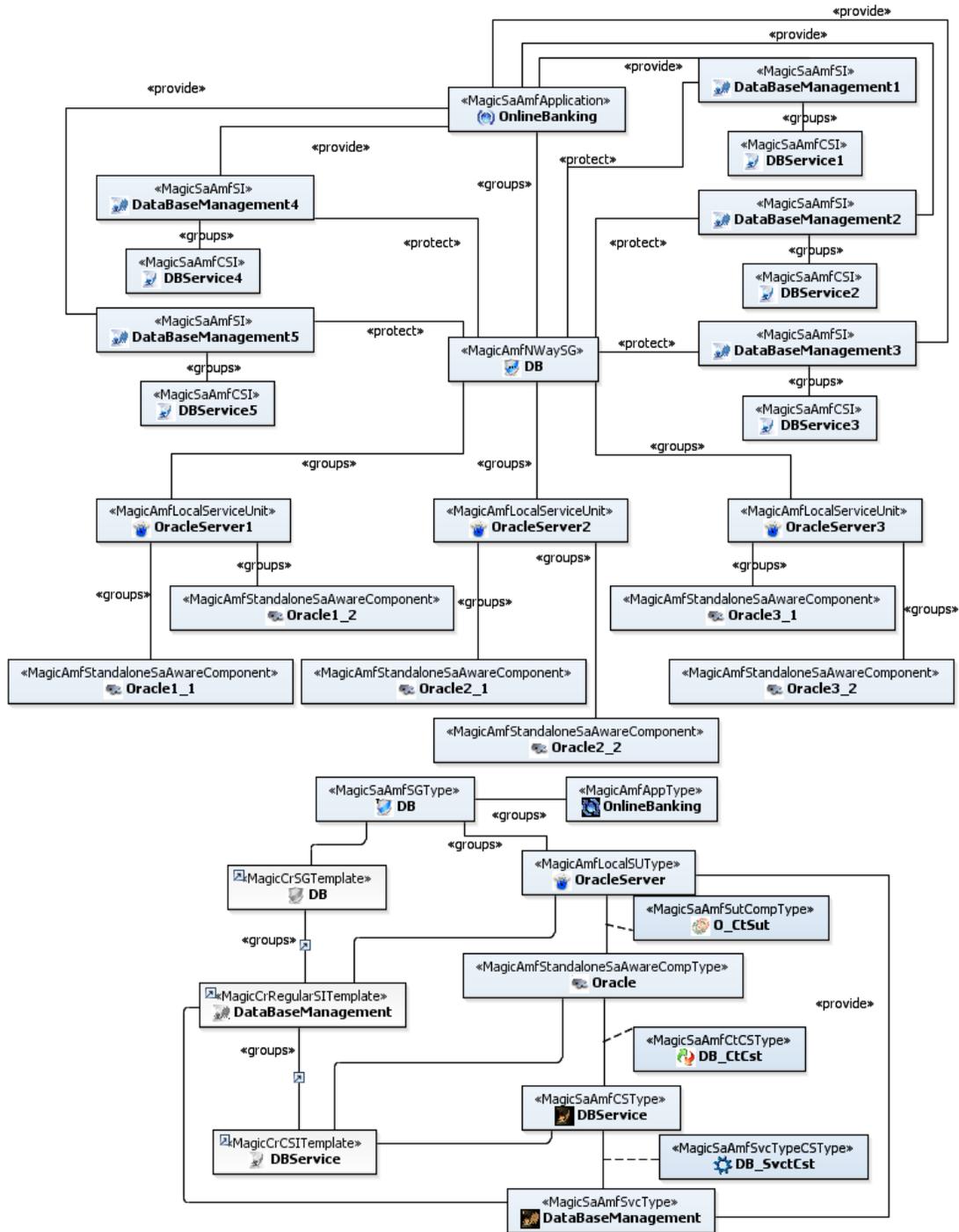


Figure 7-22 AMF entity creation phase for the DBMS part of online banking configuration

Figure 7-22 shows AMF entities instantiated for the DBMS part of the online banking system. It should be noted that the links between AMF entity types and AMF entities are omitted from this figure for readability purposes. Moreover, the elements of the CR

model will also be pruned out in the very last step of the AMF type creation phase (see Section 6.4.3).

Finally, at the deployment level, ten identical nodes are created and all SUs in the configuration are evenly distributed among these nodes. A single cluster is generated to group these nodes.

7.5 Validation of the Model-based AMF Configuration Generation Approach

The extensive usage of model transformations in the development of systems has led researchers to apply software development techniques, such as formal validation and verification as well as testing approaches on model transformations. The formal validation and verification of transformations have been studied by different research groups. Varro and Pataricza [Varro 2003] proposed a model-level automated technique to formally verify model transformations. Their approach verifies whether the transformation from a specific well-formed source model into its target equivalent preserves the dynamic consistency properties of the target metamodel. This approach is based on model checking and has practical limitations imposed by the state explosion problem.

In [Küster 2004], the author introduced a systematic approach for the validation of transformations, focusing on their syntactical correctness. This work has been continued and presented in [Küster 2006] by focusing on the formal investigation of the termination and confluence properties of model transformations, i.e. to ensure that, given a source model, a model transformation always produces a unique target model as result. Although the author presents the theoretical part of the approach that needs to be taken into

consideration by software designers, the tool support component was not presented in these works.

In a recent paper [Cabot 2010] Cabot et al. proposed verification and validation techniques for M2M transformations based on the analysis of a set of OCL invariants automatically derived from the declarative description of the transformations. These invariants state the conditions that must hold between a source and a target model in order to satisfy the transformation definition. These invariants, together with the source and target meta-models, form transformation models and were analyzed by translating them into a constraint satisfaction problem using the UMLtoCSP [Cabot 2009 and Cabot 2008] tool which is then processed with constraint solvers to verify transformations. The authors also proposed an approach for validating the transformation by generating valid pairs of source and target models using the UMLtoCSP tool. Although the presented approach provides a comprehensive technique for the validation and verification of the transformations, the tool support is limited due to the complexity of the transformation models. This results in an exponential execution time or leads to undecidable or incomplete decision problems, hindering the scalability of the approach.

There are also other works in the area of formal verification and/or validation of model transformations [Ehring 2007 and Lengyel 2010]. Similarly, these approaches also suffer from scalability issues, due to computational complexity and/or the state explosion problem. As a result, existing techniques cannot be applied on our model-based configuration generation approach which consists of a large number of transformation rules as well as complex input/output metamodels.

We believe we have followed a rigorous and stepwise process in designing the model-based approach. Reusing the knowledge gained during the specification of our modeling framework which was validated by a domain expert certainly decreased the probability errors in our approach. Indeed, for specifying the transformations rules we reused most of the OCL constraints specified in the AMF sub-profile of our modeling framework.

Designing our approach in a stepwise manner allowed us to test each step independently by defining appropriate test cases. In each step different rules capture different possible scenarios and through the appropriate definition of our test cases, we have activated the pre-conditions of each rule and have covered the various possible scenarios.

Testing is a partial validation technique that can be performed on model transformation approaches. This is a challenging activity and there is ongoing research in this field [Baudry 2006, Baudry 2010]. This process becomes even more challenging for systems involving model-based AMF configuration generation that have complex metamodels with large numbers of OCL constraints. Literature reports on the number of solutions for testing model transformations mainly follow the black box testing strategy. For instance, McGill et al. [McGill 2007] introduced an extension of the JUnit testing framework including model transformation which facilitates the definition of simple Java test cases for models represented in XML. Sen et al. [Sen 2008] presented a tool for automatic test case generation which uses Alloy language. A recent work by Ciancone et al. [Ciancone 2010] concentrates on the white box testing strategy and focuses on the testing approach for QVTO-based model transformations. The drawback of this approach is that it is tightly coupled to the QVTO [OMG 2007c] transformation language.

These approaches, however, are subject to ongoing research and mainly suffer from the absence of a mature oracle capable of handling large complex systems and metamodels [Mottu 2008]. The strategy we used for testing our approach is based on the traditional black box testing [Beizer 1995]. As specified in Chapter 6, in each of the three main phases of our approach we store the selected/created elements which can be used to test each step individually. More specifically, in each step we checked if the transformation rules generate the desired output based on a given input model. We have also tested the entire approach by considering the complete set of transformations as a black box and focused on checking if the requirements specified in the CR model were satisfied in the final generated AMF configuration. The criteria that can be checked for the generated SIs in the configuration are as follows:

- The redundancy model: For each SI whether the redundancy model of the protecting SG is compliant with the redundancy model specified in the SGTemplate of the corresponding SITemplate.
- The number of SIs created: The number of generated SIs is the same as the required number of SIs specified in the corresponding SITemplate.
- The dependency: The compliance between the dependency specified in the CR model and the dependency captured in the configuration.
- The number of CSIs created: For each SI whether the number of generated CSIs is the same as the number specified in the CSITemplates of the corresponding SITemplate.

In addition to the abovementioned strategies, we can also test the final generated configuration using the validation approach presented in Chapter 5. Although our

validation approach is designed for the validation of the third-party configurations, using this approach will assure the validity of the configuration with respect to the concepts and constraints of the standard specification and can be used as a test strategy for the model-based configuration generation.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

In this thesis, we have proposed a model-based framework for AMF configuration management. The proposed approach is based on the model driven paradigm which has been shown to result in improved quality, serviceability, portability, and flexibility. Our approach consists of a modeling framework, an AMF configuration validation approach and a model-based AMF configuration generation approach.

The modeling framework is built as a UML profile and is composed of three sub-profiles: AMF, ETF, and CR. These sub-profiles specify the concepts and semantics related to AMF configurations, the description of the software, as well as the configuration requirements, respectively.

The AMF sub-profile facilitates the design, generation, and validation of AMF configurations while the ETF sub-profile supports the design and specification of software descriptions for SA Forum compliant software.

Our approach also includes a model-based method for generating AMF configurations and another one for validating third-party AMF configurations. The model-based configuration generation approach is based on three profiles that capture elements representing different artefacts involved in the generation process. The proposed

approach is defined in terms of these artefacts and abstracts away any specific code and implementation details. This reduces the likelihood of potential errors and improves the maintainability of the solution, as opposed to a code-centric approach. More specifically, by using a model transformation technique and a declarative implementation style, future modifications of the profiles will have less impact on the implementation compared to a code-centric approach. Furthermore, the domain knowledge that has been modeled in profiles is reused directly in the model-driven approach. For instance, the well-formedness rules described in the profiles in terms of OCL constraints are used to derive the definition of the transformation rules.

Our model-driven configuration generation process is implemented using ATL, a well-known toolkit for model transformation, and is based on previously defined UML profiles. The usage of these de-facto standard technologies favours the diffusion and usability of our solution. Moreover, the proposed transformation rules can be easily integrated and executed in any UML CASE tool.

For validating third-party AMF configurations, the syntactical validation was performed by mapping these configurations to our modeling framework and checking their compliance against the AMF specification. We have also proposed an approach for the semantic validation of AMF configurations, i.e. whether a given AMF configuration provided the level of protection it claimed. To this end, we explored and discussed this issue, referred to as the SI-Protection problem, and we proved that in the case of N+M, N-Way and N-Way-Active redundancy models the problem is NP-hard in general. For these three redundancy models, we identified some specific situations where the problem can be simplified. We tackled the problem further and proposed a solution for these

redundancy models that is founded on heuristics and based on extensions to the well-known bin-packing problem. As a result, we have introduced seven different heuristic methods for checking the SI-Protection problem. To achieve better results, our approach applies all proposed methods and determines the answer based on the outcome of these methods. Finally, we devised an approach which incrementally adds resources to a “likely” invalid configuration and transforms it into a valid one.

As a final note, this doctoral research has been part of the MAGIC research project which was carried out in collaboration with Ericsson. This opportunity has provided us with a practical real world context. Our findings have been delivered to the industrial partner in the project.

8.2 Future Research

Several issues are left open in this thesis and will be summarized in the following categories.

8.2.1 Model-based AMF Configuration Generation

Our model-based configuration generation approach considers the redundancy model that should be used to protect the services. This property allows for generating AMF configurations that can support the required protection level associated with the redundancy model. This represents a first step towards the definition of a generation process that considers both functional and non-functional (NF) requirements. The proposed process could be refined considering additional NF properties belonging to the availability category, such as the level of availability, the mean time to failure, etc. Moreover, properties belonging to other categories also could be used to refine the

generation of configurations. For instance, by knowing how much a customer is allowed to invest and the cost associated with the SW bundle elements, one could generate AMF configurations whose cost complies with the budget. Another refinement could be enabled by performance properties, such as the desired response time or throughout, and the corresponding aspects of the SW bundle.

In this regard, optimizing the generated configuration according to different NF properties can also be investigated in the future. Different design decisions and/or patterns could be introduced and considered in the generation process for supporting the optimization of the designed configuration according to a specific NF property. Considering multiple NF requirements simultaneously is also a potential future research topic.

8.2.2 Performance Evaluation of Heuristics Based Validation Approach

So far, we have checked our heuristic approaches on a limited number of small scale configurations that were generated automatically by our AMF configuration generation method. However, these configurations were not appropriate for the performance analysis of the validation approach. In order to analyze the performance of the approach, it is necessary to have a set of large scale configurations. This set also needs to include a variety of configurations in order to cover different criteria such as the variation of SIs or SUs based on the number of CTypes they require/provide. Therefore, analysing such performance is a complex task which requires the implementation of a simulation framework for different scenarios. In addition, it is possible to introduce new heuristics focusing on the order of the SIs or alternative sorting criteria. Future work could involve

the investigation of this simulation framework, a thorough analysis of the performance of our approach, as well as the design of new heuristics.

8.2.3 Bridging the Gap between User Requirements and Configuration Requirements

As discussed in Chapter 6, one of the limitations of our model-based AMF configuration approach is that the CR model uses elements close to AMF configuration concepts. Specifying CR model elements requires broad domain knowledge and expertise. Therefore, there is a gap between the high level user requirements and the configuration requirements. In the future we can bridge this gap by adding an extra step for processing and refining high level requirements into configuration requirements, a step which complements our current approach. More specifically, this additional phase incorporates the specification and decomposition of the user requirements and generates the CR to be used for our current approach.

8.2.4 UML Profiling

Although UML profiling is a well-known technique for the design of DSMLs, most UML profiles were designed in an ad hoc manner, resulting in UML profiles that are either technically invalid or of poor quality. Another major shortcoming in this area is the lack of a well-defined evaluation mechanism for evaluating UML profiles. Therefore, the following issues can be addressed in the future work of this research stream:

- The design of a systematic approach to improve the process of defining UML profiles.

- The specification of a well-defined evaluation framework and metrics in order to support the formal evaluation of the UML profiles.

8.2.5 Model-driven Software Development

Model transformations that analyze certain aspects of models and then produce different types of artefacts (e.g. different models) constitute an integral part of the MDE. Despite the efforts that have been made in proposing different tools and languages to support model transformations, these tools focus primarily on the implementation phase of the software development. Therefore, the objective of another future research stream could involve the specification of a well-defined software process based on model transformation technology. This process will represent a networked sequence of activities, objects, and artefacts that embody strategies for accomplishing software evolution and will prove useful for developing more precise and formalized descriptions of software life cycle activities (e.g. analysis and design).

Related Publications

- P. Salehi, F. Khendek, M. Toeroe, A. Hamou-Lhadj, and A. Gherbi, “Checking Service Instance Protection for AMF Configurations,” in Proc. of the Third IEEE International Conference on Secure Software Integration and Reliability Improvement, Shanghai, China, IEEE Computer Society 2009, ISBN 978-0-7695-3758-0, pp. 269 - 274.
- 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, ISSN 1613-0073, pp. 56-68.
- P. Salehi, A. Hamou-Lhadj, P. Colombo, M. Toeroe, and F. Khendek, “A UML-Based Domain Specific Modeling Language for the Availability Management Framework”, in Proc. of the 12th IEEE International High Assurance Systems Engineering Symposium, San Jose, CA, IEEE Computer Society 2010, ISBN 978-1-4244-9091-2, pp. 35-44.
- 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, Lecture Notes in Computer Science 6598 Springer 2011, ISBN 978-3-642-21651-0, pp. 124-143.
- P. Salehi, F. Khendek, M. Toeroe, and, A. Hamou-Lhadj, “A Heuristic Approach on Checking Service Instance Protection for AMF Configurations”, in Proc. of

7th International Conference on Network and Service Management, Paris, France, 2011. [Acceptance Rate 15%]

- P. Salehi, A. Hamou-Lhadj M. Toeroe, P. Colombo, F. Khendek, “A Model Driven Approach for Availability Management Framework (AMF) Configuration Generation”, patent filed by Ericsson Canada, 2011.

Technical Reports Delivered to Industrial Partner

- A Model Driven Approach for Availability Management Framework Configurations Generation, 2011.
- A Heuristic Approach on Checking Service Instance Protection for AMF Configurations, 2011.
- A UML Profile for the Availability Management Framework Configurations version 2.0, 2010.
- A UML Profile for the Entity Types File version 2.0, 2010.
- A UML Profile for the Entity Types File version 1.0, 2009.
- A UML Profile for the Availability Management Framework Configurations version 1.0, 2008.

Bibliography

- [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.
- [Abouzahra 2005] A. Abouzahra, J. Bézivin, M. Didonet Del Fabro, and F. Jouault., "A practical approach to bridging domain specific languages with UML profiles," in Proc. of the Workshop on Best Practices for Model Driven Software Development, OOPSLA, San Diego, USA, 2005.
- [Amyot 2006] D. Amyot and J. Roy, "Evaluation of Development Tools for Domain-Specific Modeling Languages," In 5th International 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>.
- [Baudry 2006] B. Baudry, T. Dinh-Trong, J.M. Mottu, D. Simmonds, R. France, S. Ghosh, F. Fleurey, and Y. Le Traon, "Model transformation testing challenges," in Proceedings of IMDT workshop in conjunction with ECMDA'06, Bilbao, Spain, 2006.
- [Baudry 2010] B. Baudry, S. Ghosh, F. Fleurey, R. France, Y.L. Traon and, J.M. Mottu, "Barriers to systematic model transformation testing," Communications of the ACM 53(6), pp.139-143, (2010)
- [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 2008] S. Bernardi, J. Merseguer, and D. Petriu, "Adding dependability analysis capabilities to the MARTE profile," in Proc. of the 11th international conference on Model Driven Engineering Languages and Systems, Toulouse, France, 2008, pp. 736-750.
- [Beizer 1995] B. Beizer, "Black-box testing: techniques for functional testing of software and systems", John Wiley & Sons, Inc., New York, NY, 1995

- [Cabot 2008] J. Cabot, R. Clarisó, and D. Riera, “Verification of UML/OCL class diagrams using constraint programming,” in MoDeVVa 2008. ICST Workshop, pp. 73–80.
- [Cabot 2009] J. Cabot and E. Teniente, “Incremental integrity checking of UML/OCL conceptual schemas,” *Journal of Systems and Software* vol. 82 (9), pp. 1459-1478.
- [Cabot 2010] J. Cabot, R. Clarisó, E. Guerra, and J. de Lara, “Verification and validation of declarative model-to-model transformations through invariants,” *Journal of Systems and Software*, vol. 83, 2010, pp. 283-302.
- [Ciancone 2010] A. Ciancone, A. Filieri, and R. Mirandola, “MANTra: Towards Model Transformation Testing,” in Proc. of the Seventh International Conference on the Quality of Information and Communications Technology, Porto, Portugal, 2010, pp. 97-105.
- [Coffman 1996] E. G. Coffman, Jr. , M. R. Garey , and D. S. Johnson, “Approximation algorithms for bin packing: a survey,” *Approximation algorithms for NP-hard problems*, PWS Publishing Co., Boston, MA, 1996
- [Csirik 1990] J. Csirik, J. Frenk, M. Labbe, and S. Zhang, “On the multidimensional vector bin packing,” *European Institute for Advanced Studies in Management*, 1990.
- [Eclipse 2010a] Eclipse Foundation, 2010, URL: <http://www.eclipse.org/>
- [Eclipse 2010b] Eclipse Foundation, Eclipse Modeling Framework (EMF), 2010, URL: <http://www.eclipse.org/modeling/emf/>
- [Ehrig 2007] H. Ehrig, K. Ehrig, C. Ermel, F. Hermann, and G. Taentzer, “Information preserving bidirectional model transformations,” in Proc. of FASE’07, 2007, vol. 4422, LNCS, Springer, pp. 72-86.
- [Felfernig 2000] A. Felfernig, G. Friedrich, and D. Jannach, “UML as domain specific language for the construction of knowledge-based configuration systems,” *International Journal of Software Engineering and Knowledge Engineering*, 2000. 10(4): pp. 449-470.
- [France 2007] R. France, and B. Rumpe, “Model-driven development of complex software: A research roadmap,” in Proc. of Future of Software Engineering, Washington, DC, USA, 2007, pp. 37-54.
- [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.

- [Garey 1979] M. Garey, and D. Johnson, "Computers and intractability. A guide to the theory of NP-completeness," A Series of Books in the Mathematical Sciences. 1979: WH Freeman and Company, San Francisco, CA, USA.
- [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.
- [Gray 2006] J. Gray, Y. Lin, and J. Zhang, "Automating change evolution in model-driven engineering," Computer, v.39 n.2, pp.51-58, 2006.
- [IBM 2011] IBM Rational Software Architect (RSA), <http://www-01.ibm.com/software/awdtools/architect/swarchitect/>
- [Kanso 2008] A. Kanso, A. Hamou-Lhadj, M. Toeroe, and F. Khendek, "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, A. Hamou-Lhadj, M. Toeroe, 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.
- [Kelly 2008] S. Kelly, and J. Tolvanen, "Domain-specific modeling: enabling full code generation," Wiley-IEEE Computer Society Press, 2008.
- [Kenyon 1996] C. Kenyon, "Best-fit bin-packing with random order," in Proc. of the seventh annual ACM-SIAM symposium on Discrete algorithms, p.359-364, January 28-30, 1996, Atlanta, Georgia, United States
- [Knapp 2003] A. Knapp, N. Koch, F. Moser, and G. Zhang, "ArgoUWE: A Case Tool for Web Applications," in Proc. of the First Int. Workshop on Engineering Methods to Support Information System Evolution, Geneva, Switzerland, 2003.
- [Kövi 2007] A. Kövi, "UML profile and design patterns library. (Preliminary version)," Aalborg University, Aalborg, Denmark, IST-FP6-STREP-26979 / HIDENETS, 2007.
- [Küster 2004] J.M. Küster, "Systematic Validation of Model Transformations," in the 3rd UML Workshop in Software Model Engineering (WiSME 2004), <http://www.metamodel.com/wisme-2004/accept/4.pdf>.
- [Küster 2006] J.M. Küster, "Definition and validation of model transformations," Software and Systems Modeling, Volume 5, Number 3, 2006, pp.

233-259.

- [Lagarde 2007] F. Lagarde, H. Espinoza, F. Terrier, and S. Gérard, “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, H. Espinoza, F. Terrier, C. André, and S. Gérard, “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.
- [Lengyel 2010] L. Lengyel, I. Madari, M. Asztalos, and T. Levendovszky,” Validating Query/View/Transformation Relations,” in Proc. of 2010 Workshop on Model-Driven Engineering, Verification, and Validation, 2010, Oslo, Norway, pp. 7-12.
- [Leroux 2006] D. Leroux, M. Nally and K. Hussey “Rational Software Architect: A tool for domain-specific modeling,” IBM System Journal, 2006.
- [Lomb 1996] R. Lomb, K. Emo, and R. VanDoorn, “Storage management solutions for distributed computing environments.” HEWLETT PACKARD JOURNAL, 1996. 47: pp. 81-93.
- [McGill 2007] M. J. McGill and B. H. C. Cheng, “Test-driven development of a model transformation with jemtte,” Technical Report, Software Engineering and Network Systems Laboratory, Department of Computer Science and Engineering, Michigan State University, 2007.
- [Mernik 2005] M. Mernik, J. Heering, and A. Sloane, “When and how to develop domain-specific languages,” ACM Computing Surveys (CSUR), 2005. 37(4): pp. 316-344.
- [Motto 2008] J.M. Motto, B. Baudry, and Y.L. Traon, “Model transformation testing: Oracle issue,” In: Proc. of MoDeVVA workshop colocated with ICST 2008, Lillehammer, Norway (April 2008)
- [OMG 2002] Object Management Group, UML Profile for CORBA Specification, Version 1.0, formal/02-04-01, URL: <http://www.omg.org/cgi-bin/doc>.
- [OMG 2003a] Object Management Group, UML Profile for Schedulability, Performance, and Time Specification, Version 1.1, formal/03-09-01, 2003, URL: <http://www.omg.org/cgi-bin/doc?formal/03-09-01>.

- [OMG 2003b] Object Management Group, Common Warehouse Metamodel (CWM™) Specification, Version 1.1, formal/2003-03-02, 2003, URL: <http://www.omg.org/spec/CWM/1.1/>.
- [OMG 2004] UML 2.0 Testing Profile Specification, Version 1.0, ptc/2004-04-02, 2004, URL: <http://www.omg.org/cgi-bin/doc?ptc/2004-04-02>.
- [OMG 2006a] Object Management Group, Meta Object Facility (MOF) Core Specification, Version 2.0, formal/2006-01-01, 2006, URL: <http://www.omg.org/spec/MOF/2.0>.
- [OMG 2006b] AUTOSAR GbR, UML Profile for AUTOSAR Specification, Version 1.0.1. 2006, URL: <http://www.autosar.org>.
- [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 2007c] Object Management Group. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. Final Adopted Specification, ptc/07-07-07, 2007, [URL:http://www.omg.org/cgi-bin/doc?ptc/2007-07-07](http://www.omg.org/cgi-bin/doc?ptc/2007-07-07)
- [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 2009] Object Management Group, A UML Profile for MARTE Specification, Version 1.0, formal/2009-11-02, 2009, URL: <http://www.omg.org/spec/MARTE/index.htm>.
- [OMG 2010a] OMG, Object Constraint Language, Version 2.2 - <http://www.omg.org/spec/OCL/2.2/PDF>
- [OMG 2010b] Object Management Group, SysML Specification, Version 1.2 formal/10-06-02, 2010, URL: <http://www.sysml.org/specs.htm>.
- [OMG 2011] Object Management Group, URL: <http://www.omg.org>
- [Patt-Shamir 2010] B. Patt-Shamir, and D. Rawitz, "Vector Bin Packing with Multiple-Choice," in Proc. of the 12th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT), Bergen, Norway, 2010, pp. 248-259.

- [Pearl 1984] J. Pearl, "Heuristics: Intelligent Search Strategies for Computer Problem Solving," New York, Addison-Wesley, 1984.
- [Rao 2010] C.S. Rao, J.J. Geevarghese, and K. Rajan., "Improved Approximation Bounds for Vector Bin Packing," Arxiv preprint arXiv:1007.1345, 2010.
- [SAF 2010a] Service Availability Forum™, URL: <http://www.saforum.org>
- [SAF 2010b] Service Availability Forum™, Overview SAI-Overview-B.05.03 at: http://www.saforum.org/link/linkshow.asp?link_id=222259&assn_id=16627
- [SAF 2010c] 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 2010d] Service Availability Forum™, Application Interface Specification. Availability Management Framework SAI-AIS-AMF-B.04.01
- [SAF 2010e] Service Availability Forum, Application Interface Specification. Software Management Framework SAI-AIS-SMF-A.01.01.
- [Salehi 2009] P. Salehi, F. Khendek, M. Toeroe, A. Hamou-Lhadj, and A. Gherbi, "Checking Service Instance Protection for AMF Configurations," in Proc. of the Third IEEE International Conference on Secure Software Integration and Reliability Improvement, Shanghai, China, 2009, pp. 269 - 274.
- [Salehi 2010a] P. Salehi, A. Hamou-Lhadj, P. Colombo, M. Toeroe, and F. Khendek, "A UML-Based Domain Specific Modeling Language for the Availability Management Framework," in Proc. of The 12th IEEE International High Assurance Systems Engineering Symposium, San Jose, CA, 2010, pp. 35-44.
- [Salehi 2010b] 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 2011a] P. Salehi, F. Khendek, M. Toeroe, and A. Hamou-Lhadj "A Heuristic Approach on Checking Service Instance Protection for AMF Configurations," in Proc. of 7th International Conference on Network and Service Management, Paris, France, 2011.
- [Salehi 2011b] P. Salehi, A. Hamou-Lhadj M. Toeroe, P. Colombo, F. Khendek, "A Model Driven Approach For Availability Management

- Framework (AMF) Configuration Generation,” patent application filed by Ericsson Canada, 2011.
- [Salehi 2011c] P. Salehi, A. Hamou-Lhadj, M. Toeroe, and F. Khendek, “A Precise UML Domain Specific Modeling Language for Service Availability Management,” submitted to the Journal of Systems and Software 2011.
- [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.
- [Sen 2008] S. Sen, B. Baudry, and J. M. Mottu, “On combining multi-formalism knowledge to select models for model transformation testing,” in Proc. Of the 1st International Conference on Software Testing, Verification, and Validation, Lillehammer, Norway, 2008, pp. 328-337.
- [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.
- [Varro 2003] D. Varro and A. Pataricza, “Automated formal verification of model transformations,” in Proc. of the UML'03 Workshop, Number TUM-I0323 in Technical Report, Technische Universität München, 2003 pp. 63-78.
- [VideoLAN 2010] VideoLAN Project, The VideoLan Server (VLS) System, 2010, URL: <http://www.videolan.org/>
- [Vogels 1998] W. Vogels, “The Design and Architecture of the Microsoft Cluster Service-A Practical Approach to High- Availability and Scalability,” in Proc. of 28th Symposium on Fault-Tolerant Computing, CS Press, 1998, pp. 422-431.
- [Wang 2005] D. Wang, and K. Trivedi, “Modeling user-perceived service availability,” in Service Availability, LNCS, 2005, Volume 3694/2005, pp. 107-122.
- [Watts 2007] D. Watts, R.J. Brenneman, D. Feisthammel, and T. Sutherland, “Implementing IBM Director 5.20,” IBM Redbooks, April, 2007.

Appendix I:

List of the Tagged Definitions

AMF Sub-profile Tagged Definitions

Tagged Definition	Description
MagicSaAmfCompGlobalAttributes	
magicSafRdn	This attribute contains the name of the object of this class
magicSaAmfNumMaxInstantiateWithoutDelay	This attribute specifies the maximum number of unsuccessful instantiation attempts without delay performed consecutively by AMF
magicSaAmfNumMaxInstantiateWithDelay	This attribute indicates the maximum number of attempts that AMF can make to instantiate the component with a delay between the attempts
magicSaAmfNumMaxAmStartAttempts	The value of this attribute is the maximum number of attempts to start the active monitoring
magicSaAmfNumMaxAmStopAttempts	The value of this attribute is the maximum number of attempts to stop the active monitoring of the component
magicSaAmfDelayBetweenInstantiateAttempts	This value is the delay period that AMF waits before the next attempt to instantiate a component after failing to instantiate it
SaAmfCompBaseType	
safCompType	This attribute contains the name of this base type
MagicSaAmfCompType	
magicSafVersion	value of this attribute is the version of the

	component type
magicSaAmfCtDefRecoveryOnError	This attribute specifies the recovery action that should be taken by AMF by default for the components of this type
magicSaAmfCtDefDisableRestart	The value of this attribute indicates whether the restart recovery action is disabled or not by default for the components of this type
magicSaAmfCtDefClcCliTimeout	The value of this attribute is the default value for the time that the process of executing of a CLC-CLI command for the components of this type should not exceed otherwise the execution of the command fails
magicSaAmfCtDefAmStartCmdArgv	This attribute defines the default arguments for the CLC-CLI command used to start the active monitoring of a component of this type
magicSaAmfCtDefStopCmdArgv	This attribute defines the default arguments for the CLC-CLI command used to stop the active monitoring of a component of this type
magicSaAmfCtRelPathAmStartCmd	This attribute denotes the relative pathname of the AM-START CLC-CLI command of components of this type
magicSaAmfCtRelPathAmStopCmd	This attribute defines the relative pathname of the AM-STOP CLC-CLI command for the components of this type
magicSaAmfCtDefCallbackTimeout	This attribute defines the default value for all callback timeouts of the components of this type. This value will be used for all callback timeouts that are not specified for such a component
MagicAmfSaAwareCompType	
magicSaAmfCtDefInstantiationLevel	This attribute defines the default value for instantiation level of the components of this type
magicSaAmfCtDefQuiescingCompleteTimeout	The value of this attribute defines the default time limit used at quiescing of the CSIs assigned to components of this type
MagicAmfStandaloneSaAwareCompType	
magicSaAmfCtDefCleanupCmdArgv	This attribute defines the default

	arguments for the CLC-CLI cleanup command for all components of this type
magicSaAmfCtRelPathAmfCleanupCmd	This attribute defines the relative path for the cleanup command of the components of this type
magicSaAmfCtDefInstantiateCmdArgv	This attribute defines the default arguments for the CLC-CLI instantiate command used for the components of this type
magicSaAmfCtRelPathAmfInstantiateCmd	This attribute defines the relative path for the instantiate command of the components of this type
magicSaAmfCtDefCmdEnv	This attribute defines the default environment variables and their values for all CLC-CLIs commands of the components of this type
MagicAmfStandaloneSaAwareCompType	
magicSaAmfCtDefCleanupCmdArgv	This attribute defines the default arguments for the CLC-CLI cleanup command for all components of this type
magicSaAmfCtRelPathAmfCleanupCmd	This attribute defines the relative path for the cleanup command of the components of this type
magicSaAmfCtDefInstantiateCmdArgv	This attribute defines the default arguments for the CLC-CLI instantiate command used for the components of this type
magicSaAmfCtRelPathAmfInstantiateCmd	This attribute defines the relative path for the instantiate command of the components of this type
magicSaAmfCtDefCmdEnv	This attribute defines the default environment variables and their values for all CLC-CLIs commands of the components of this type
MagicAmfProxiedCompType	
magicSaAmfCtDefInstantiationLevel	This attribute defines the default value for instantiation level of the components of this type
magicSaAmfCtDefQuiescingCompleteTimeout	The value of this attribute defines the default time limit used at quiescing of the CSIs assigned to components of this type
magicSaAmfCtDefCleanupCmdArgv	This attribute defines, for all the

	components of this type, the default arguments for the CLEANUP CLC-CLI command
magicSaAmfCtRelPathAmfCleanupCmd	This attribute defines the relative path for the cleanup command
magicAmfCtIsPreinstantiable	The value of this attribute indicates whether the components of this type are pre-instantiable or not
magicSaAmfCtDefCmdEnv	This attribute defines the default environment variables and their values for all CLC-CLIs commands of the components of this type
MagicAmfNon-ProxiedNon-SaAwareCompType	
magicSaAmfCtDefTerminateCmdArgv	This attribute defines, for components of this type, the default arguments for the TERMINATE CLC-CLI command
magicSaAmfCtRelPathTerminateCmd	This attribute defines the relative path for the TERMINATE CLC-CLI command which is used for the components of this type
magicSaAmfCtDefInstantiateCmdArgv	This attribute defines, for all the components of this type, the default arguments for INSTANTIATE CLC-CLI
magicSaAmfCtRelPathInstantiateCmd	This attribute defines the relative path for the INSTANTIATE CLC-CLI command which is used for the components of this type
magicSaAmfCtDefCleanupCmdArgv	This attribute defines, for all the components of this type, the default arguments for CLEANUP CLC-CLI command
magicSaAmfCtRelPathCleanupCmd	This attribute defines the relative path for the CLEANUP CLC-CLI command which is used for the components of this type
magicSaAmfCtDefCmdEnv	This attribute defines the default environment variables and their values for all CLC-CLIs commands of the components of this type
MagicSaAmfHealthcheckType	
magicSaAmfHctDefPeriod	This attribute defines the default time interval at which the health check is performed
magicSaAmfHctDefMaxDuration	This attribute defines the period during

	which AMF expects a response to the health check callback from a component of the component type associated with this health check type
SaAmfSUBaseType	
safSuType	The name of the base service unit type
MagicSaAmfSUType	
magicSafVersion	This attribute defines the version of the service unit type
magicSaAmfSutDefSUFailover	This attribute specifies whether the fail-over recovery is done for an entire service unit of this type or not
SaAmfSGBaseType	
safSgType	This attribute defines the name of the service group base type
MagicSaAmfSGType	
magicSafVersion	attribute defines the version of the service group type
magicSaAmfSgtRedundancyModel	This attribute specifies the redundancy model of the service group type
magicSaAmfSgtDefAutoAdjust	This attribute defines the default value of the MagicSaAmfSG::magicSaAmfSGAutoAdjust attribute for all service groups of this type, which indicates whether the auto adjust operation is enabled or not
magicSaAmfSgtDefAutoRepair	This attribute defines the default value of MagicSaAmfSG::magicSaAmfSGAutoRepair attribute for all service groups of this type, which specifies whether the Availability Management Framework engages in automatic repair or not at service group level.
magicSaAmfSgtDefAutoAdjustProb	This attribute defines the default value of the MagicSaAmfSG::magicSaAmfSGAutoAdjustProb which defines the auto adjust probation period. This period indicates the

	time during which a service unit belonging to a service group of this type may not participate in an auto-adjust procedure. After this period it becomes eligible for assignments as part of an auto-adjustment executed as a consequence of a repair/recovery action
magicSaAmfSgtDefCompRestartProb	This attribute defines the default value of the MagicSaAmfSG::magicSaAmfSGCompRestartProb which specifies the component restart probation period.
magicSaAmfSgtDefCompRestartMax	This attribute defines the default value for MagicSaAmfSG::magicSaAmfSGCompRestartMax which is the maximum number of components of any service unit in a service group of this type that can be restarted within the component restart probation time without triggering a first level escalation
magicSaAmfSgtDefSuRestartProb	This attribute defines the default value for MagicSaAmfSG::magicSaAmfSGSuRestartProb which is the restart probation period of a service unit in a service group of this type
magicSaAmfSgtDefSuRestartMax	This attribute is the default value for MagicSaAmfSG::magicSaAmfSGSuRestartMax which is the maximum number a service unit in a service group of this type can be restarted without causing a SU failover
SaAmfAppBaseType	
safAppType	This attribute specifies the name of the application base type
MagicAmfAppType	
magicSafVersion	This attribute specifies the version for the application type
MagicSaAmfSutCompType	
magicSaAmfSutMaxNumComponents	This attribute specifies the maximum number of components of the associated component type that can be members of a

	service unit from the related service unit type
magicSaAmfSutMinNumComponents	This attribute specifies the minimum number of components of the associated component type that must be members of a service unit from the related service unit type
SaAmfCSBaseType	
safCSType	This attribute specifies the name of the component service instance base type
MagicSaAmfCSType	
magicSafVersion	This attribute specifies the version of component service instance type
SaAmfSvcBaseType	
safSvcType	attribute defines the name of the service base type
MagicSaAmfSvcType	
magicSafVersion	This attribute specifies the version of the service type
magicSaAmfSvcDefActivWeight	This attribute represents the default value for the load that service instances of this service type will impose on the node when assigned to a service unit of the node as active, quiescing, or quiesced
magicSaAmfSvcDefStandbyWeight	This attribute represents the default value for the load that service instances of this service type will impose on the node when assigned to a service unit of the node as standby
MagicSaAmfSvcTypeCSType	
magicSaAmfSvctMaxNumCSIs	The value of this attribute indicates the maximum number of CSIs of the associated CStype (identified by magicSafMemberCSType) can be in a service instance of the service type

MagicSaAmfCtCSType	
magicSaAmfCtCompCapability	This attribute defines the component capability model of the components of the component type with respect to the CSI of the CSType
magicSaAmfCtDefNumMaxActiveCSIs	This attribute defines the maximum number of active assignment CSIs of the CSType to the components of the component type
magicSaAmfCtDefNumMaxStandbyCSIs	This attribute defines the maximum number of standby CSIs of the CSType that can be assigned to the components of the component type
MagicSaAmfComp	
magicSafComp	This attribute contains the relative distinguished name of a component
magicSaAmfCompDisableRestart	This contains a Boolean value which determines the applicable presence state model at component failure
magicSaAmfCompRecoveryOnError	This attribute specifies the recovery action that should be taken by AMF for the component
magicSaAmfCompInstantiateTimeout	The value of this attribute is the time that the instantiation of the component should not exceed otherwise the instantiation of the component fails
magicSaAmfCompCleanupTimeout	The value of this attribute is the time that the process of cleaning up the component should not exceed otherwise the termination of the component fails
magicSaAmfCompNumMaxInstantiateWithoutDelay	This attribute indicates the number of attempts that AMF can make to instantiate the component without delay between the attempts
magicSaAmfCompNumMaxInstantiateWithDelay	attribute indicates the number of attempts that AMF can make to instantiate the component with a delay between the attempts
magicSaAmfCompDelayBetweenInstantiateAttempts	The value of this attribute indicates the delay between instantiation attempts
magicSaAmfCompTerminateTimeout	The value of this attribute is the time that the termination of a component should not exceed otherwise AMF will attempt the cleanup of the component

MagicAmfLocalComponent	
magicSaAmfCompAmStartCmdArgv	This attribute contains additional arguments for the CLC-CLI command, which is used to start the active monitoring for the component
magicSaAmfCompAmStartTimeout	The value of this attribute is the time that starting the active monitoring of the component should not exceed
magicSaAmfCompNumMaxAmStartAttempts	The value of this attribute is the maximum number of attempts to start the active monitoring
magicSaAmfCompAmStopCmdArgv	This attribute contains additional arguments for the CLC-CLI command that is used to stop active monitoring of the component
magicSaAmfCompAmStopTimeout	This value of this attribute is the time that the completion of the command for stopping active monitoring of the component should not exceed
magicSaAmfCompNumMaxAmStopAttempts	The value of this attribute is the maximum number of attempts to stop the active monitoring of the component
MagicAmfExternalComponent	
magicSaAmfCompInstantiationLevel	This attribute reflects the order in which components are instantiated within the service unit. Components having a lower instantiation level must be instantiated prior to components having a higher instantiation level
magicSaAmfCompCSISetCallbackTimeout	The value of this attribute represents the time limit for setting the HA state of the component on behalf of some CSI
magicSaAmfCompCSIRmvCallbackTimeout	The value of this attribute is the time limit for removing one or all component service instances from the set of component service instances assigned to the component
magicSaAmfCompQuiescingCompleteTimeout	The value of this attribute represents the time limit for the component to complete the process of quiescing
MagicAmfSaAwareComponent	

magicSaAmfCompInstantiationLevel	The value of this attribute represents the instantiation level of the component. The instantiation level reflects the order in which the components are instantiated: components with a lower instantiation level are instantiated prior to components with a higher instantiation level
magicSaAmfCompCSISetCallbackTimeout	The value of this attribute is the time limit for setting of the HA state of the component for component service instances
magicSaAmfCompCSIRmvCallbackTimeout	The value of this attribute is the time that the removal of CSI assignments from this component should not exceed
magicSaAmfCompQuiescingCompleteTimeout	The value of this attribute is the time limit that the process of quiescing of this component for component service instances assigned to it should not exceed
MagicAmfNon-SaAwareComponent	
magicSaAmfCompCleanupCmdArgv	This attribute contains any additional arguments for the CLEANUP CLC-CLI command specified in the type
magicSaAmfCompCmdEnv	This attribute defines the environment variables and their values for all CLC-CLIs commands of this component
MagicAmfStandaloneSaAwareComponent	
magicSaAmfCompCleanupCmdArgv	This attribute contains any additional arguments for the CLEANUP CLC-CLI command specified in the type
magicSaAmfCompInstantiateCmdArgv	This attribute contains the additional arguments that should be passed to the INSTANTIATE CLC-CLI command specified in the type
magicSaAmfCompCmdEnv	This attribute defines the environment variables and their values for all CLC-CLIs commands of this component
MagicAmfLocalProxiedComponent	
magicSaAmfCompInstantiationLevel	The value of this attribute represents the instantiation level of a component

magicSaAmfCompCSISetCallbackTimeout	The value of this attribute is the time limit for the setting of the HA-state of the component on behalf of component service instances assigned to it
magicSaAmfCompCSIRmvCallbackTimeout	The value of this attribute is the time that the removal of component service instances from the component should not exceed
magicSaAmfCompQuiescingCompleteTimeout	The value of this attribute is the time limit for quiescing of the component service instances assigned to this component
MagicAmfNon-ProxiedNon-SaAwareComponent	
magicSaAmfCompInstantiateCmdArgv	This attribute contains the arguments that are used by AMF to instantiate this component using the INSTANTIATE CLC-CLI command
magicSaAmfCompTerminateCmdArgv	This attribute contains the arguments that AMF uses to terminate the component using the TERMINATE CLC-CLI command
MagicSaAmfHealthcheck	
magicSaAmfHealthcheckPeriod	This attribute indicates the period at which the corresponding healthcheck should be initiated
magicSaAmfHealthcheckMaxDuration	This attribute indicates the time-limit after which the AMF will report an error on the component if no response for a healthcheck is received by the AMF
MagicSaAmfSU	
magicSafSu	This attribute contains the name of a service unit
magicSaAmfSURank	The value of this attribute is the rank of the SU within the service group
magicSaAmfSUFailover	The value of this Boolean attribute indicates whether the failure of a component of the service unit should trigger a fail-over of the entire service unit or only of the erroneous component
magicSaAmfSUMaintenanceCampaign	This attribute is used to disable the auto-repair behavior of AMF in certain

	situations
magicSaAmfSUAdminState	This attribute holds the administrative state of the service unit (this is persistent runtime attribute in the standard AMF model)
MagicSaAmfSG	
magicSafSg	This attribute contains the name of a service group
magicSaAmfSGAutoRepair	This attribute applies to any service unit of the particular service group and it indicates whether the AMF engages in automatic repair or not
magicSaAmfSGAutoAdjust	This attribute indicates that it is required that the SI assignments are transferred back to the preferred SUs as soon as possible
magicSaAmfSGNumPrefInserviceSUs	The value of this attribute is the preferred number of in-service service units
magicSaAmfSGAutoAdjustProb	The value of this attribute defines the auto-adjust probation time. It is used as follows: When a service unit becomes available after a repair/recovery operation, the service unit enters its auto-adjust probation period, during which it cannot be used for auto-adjustment
magicSaAmfSGCompRestartProb	The value of this attribute is the component restart probation period for a service unit
magicSaAmfSGCompRestartMax	The value of this attribute is the maximum number of components of a service unit that can be restarted before the end of component restart probation period without restarting the service unit. If this maximum is reached, AMF escalates the recovery action to restarting the entire service unit
magicSaAmfSGSuRestartProb	The value of this attribute is the service unit restart probation period
magicSaAmfSGSuRestartMax	The value of this attribute is the maximum number of level 1 escalation (i.e. restarting the entire service unit) that can be done within the service unit restart probation period. If this number is reached before the end of the period, then AMF would engage

	the level 2 of escalation for the service unit which is failing over the entire service unit
magicSaAmfSGAdminState	value of this attribute is the administrative state of a service group
MagicAmfNPlusMSG	
magicSaAmfSGNumPrefActiveSUs	This attribute indicates the preferred number of active service units at any time
magicSaAmfSGNumPrefStandbySUs	This attribute indicates the preferred number of standby service units at any time
magicSaAmfSGMaxActiveSIsperSU	This attribute indicates the maximum number of SIs that can be assigned as active to a service unit
magicSaAmfSGMaxStandbySIsperSU	This attribute indicates the maximum number of SIs that can be assigned as standbys to a service unit
MagicAmfNWaySG	
magicSaAmfSGNumPrefAssignedSUs	This attribute indicates the preferred number of assigned service units at any time
magicSaAmfSGMaxActiveSIsperSU	This attribute indicates the maximum number of SIs that can be assigned as active to a service unit
magicSaAmfSGMaxStandbySIsperSU	This attribute indicates the maximum number of SIs that can be assigned as standbys to a service unit
MagicAmfNWayActiveSG	
magicSaAmfSGNumPrefAssignedSUs	This attribute indicates the preferred number of assigned service units at any time
magicSaAmfSGMaxActiveSIsperSU	This attribute indicates the maximum number of SIs that can be assigned as active to a service unit
MagicSaAmfApplication	
magicSafApp	This attribute contains the name of the application
magicSaAmfApplicationAdminState	This attribute contains the administrative state of an application

MagicSaAmfCSI	
magicSafCsi	attribute contains the name of the CSI
MagicSaAmfSI	
magicSafSi	This attribute defines the name of the service instance
magicSaAmfSIRank	The value of this attribute represents the SI rank, AMF uses this rank to choose the SIs that will be supported with less than the wanted redundancy or that will be dropped completely if the set of in-service service units does not allow for the full support of all SIs
magicSaAmfSIAdminState	This attribute contains the administrative state for the service unit
magicSaAmfSIActiveWeight	This attribute represents the load that this service instance will impose on the node when assigned to a service unit of the node as active, quiescing, or quiesced
magicSaAmfSIStandbyWeight	This attribute represents the load that this service instance will impose on the node when assigned to a service unit of the node as standby
MagicSaAmfSIDependency	
magicSaAmfToleranceTime	This attribute specifies the time limit for which the dependent SI can tolerate the unassigned state of the SI on which it depends
MagicAmfCSIAttribute	
magicSaAmfCSIAttriValue	This attribute contains the values for the attribute for a particular CSI
MagicAmfCSIAttributeName	
magicSaCsiAttr	This attribute contains the name of the attribute
MagicSaAmfNode	
magicSafAmfNode	This attribute specifies the name of the

	node
magicSaAmfSuFailOverProb	This attribute defines the service unit fail-over probation period
magicSaAmfSuFailoverMax	This attribute defines the maximum number of failovers for the SUs within the failover probation period without causing a node failover
magicSaAmfAutoRepair	This attribute indicates whether the AMF engages in automatic repair or not. This attribute applies to any SU that is on this node
magicSaAmfFailfastOnTerminationFailure	This attribute indicates if AMF should engage in the node failfast recovery action when AMF fails to cleanup a component after the termination failure of the component
magicSaAmfFailfastOnInstantiationFailure	This attribute indicates whether AMF engages in the node failfast recovery action after a component instantiation failure occurs
magicSaAmfNodeAdminState	This attribute contains the administrative state of the node
magicSaAmfNodeCapacity	This attribute contains the configuration attribute which represents the capacity of the node
MagicSaAmfNodeGroup	
magicSafAmfNodeGroup	This attribute represents the name of the node group
MagicSaAmfCluster	
magicSafAmfCluster	This attribute specifies the name of the cluster
magicSaAmfClusterStartupTimeout	This attribute specifies the time from the cluster start which AMF should wait before it starts instantiating SUs and assigning SIs
magicSaAmfClusterAdminState	This attribute holds the administrative state of the cluster
MagicSaAmfNodeSwBundle	
magicSaAmfNodeSwBundlePathPrefix	This attribute specifies the path prefix which is configured for a software bundle

	regarding a specific node
MagicSaSmfSwBundle	
magicSafBundle	This attribute contains the name of software bundle
MagicSaAmfSIRankedSU	
magicSaAmfRank	This attribute specifies the rank of the SU with respect to the Service Instance
MagicAmfPrefActiveAssignment	
magicSaAmfSIPrefActiveAssignments	This attribute defines the preferred number of service units that are assigned the active HA state for a SI within the protecting service group, which must be of MagicNWayActiveSG
MagicAmfPrefStandbyAssignment	
magicSaAmfSIPrefStandbyAssignments	This attribute defines the preferred number of service units that are assigned the standby HA state for this SI within the protecting service group, which must be of MagicNWaySG
MagicSaAmfCompCsType	
magicSaAmfCompNumMaxActiveCSIs	This attribute specifies the maximum number of active CSIs of the CSType that can be assigned to the associated component
magicSaAmfCompNumMaxStandbyCSIs	This attribute specifies the maximum number of standby CSIs of the CSType that can be assigned to the associated component

ETF Sub-profile Tagged Definitions

Tagged Definition	Description
MagicEtfCompBaseType	
magicEtfCtName	This attribute specifies the name of the component base type
MagicEtfCompType	
magicEtfCtVersion	This attribute specifies the version for the component type
magicEtfCtDisableRestart	The value of this attribute indicates whether the software implementation is able to perform restart recovery action or not
magicEtfCtRecoveryOnError	This attribute specifies the recovery action recommended by the vendor
magicEtfCtClcCliTimeout	This attribute contains the lower bound and a possible default value for the CLC-CLI commands
magicEtfCtCallbackTimeout	This attribute defines the lower bound and a possible default value for all callback timeouts. This attribute specifies time for the callbacks if the implementation imposes any restriction. If there is a restriction, the vendor needs to provide the minimum timeout that AMF shall use for the callbacks
magicEtfCtAmStartCmd	This attribute contains the AM-START CLC-CLI command string which also includes the relative path of the command and needs to be adjusted to the execution environment
magicEtfCtAmStartCmdArgv	This attribute contains arguments of the AM-START CLC-CLI command ,which needs to be adjusted to the execution

	environment
magicEtfCtAmStopCmd	This attribute contains the AM-STOP CLC-CLI command string, which also includes the relative path of the command and needs to be adjusted to the execution environment
magicEtfCtAmStopCmdArgv	This attribute contains arguments of the AM-STOP CLC-CLI command and needs to be adjusted to the execution environment
MagicEtfSaAwareCompType	
magicSaAmfCtDefInstantiationLevel	This attribute contains minimum timeout and possible default timeout for the quiescing complete callback timeout
MagicEtfNonProxiedNonSaAwareCompType	
magicEtfCtInstantiateCmd	This attribute contains the INSTANTIATE CLC-CLI command string, which includes the path relative to the installation location for the command and needs to be adjusted to the execution environment
magicEtfCtInstantiateCmdArgv	This attribute contains the arguments of the INSTANTIATE CLC-CLI command and needs to be adjusted to the execution environment
magicEtfCtTerminateCmd	This attribute contains the TERMINATE CLC-CLI command string which also includes the path relative to the installation location for the command. It needs to be adjusted to the execution environment
magicEtfCtTerminateCmdArgv	This attribute contains arguments of the TERMINATE CLC-CLI command and needs to be adjusted to the execution environment
magicEtfCtCleanupCmd	This attribute contains the CLEANUP CLC-CLI command

	string which includes the path relative to the installation location for the command and needs to be adjusted to the execution environment
magicEtfCtCleanupCmdArgv	This attribute contains arguments of the CLEANUP CLC-CLI command and needs to be adjusted to the execution environment
MagicEtfProxiedCompType	
magicEtfCtCleanupCmd	This attribute contains the CLEANUP CLC-CLI command which string includes the path relative to the installation location for the command. It needs to be adjusted to the execution environment
magicEtfCtCleanupCmdArgv	This attribute contains arguments of the CLEANUP CLC-CLI command and needs to be adjusted to the execution environment
magicEtfCtQuiescingCompleteTimeout	This attribute contains the minimum and any recommended default timeout value for the quiescing complete callback timeout
magicEtfCtIsPreinstantiable	This attribute specifies whether the component type is pre- instantiable or not. In other words this attribute indicates whether the component type is capable of being standby or not. Non-preinstantiable components cannot act as spare nor be Idle
MagicEtfIndependentCompType	
magicEtfCtInstantiateCmd	This attribute contains the INSTANTIATE CLC-CLI command string, which includes the path relative to the installation location for the command and needs to be adjusted to the execution environment
magicEtfCtInstantiateCmdArgv	This attribute contains the

	arguments of the INSTANTIATE CLC-CLI command and needs to be adjusted to the execution environment
magicEtfCtCleanupCmd	This attribute contains the CLEANUP CLC-CLI command string which includes the path relative to the installation location for the command and needs to be adjusted to the execution environment
magicEtfCtCleanupCmdArgv	This attribute contains arguments of the CLEANUP CLC-CLI command and needs to be adjusted to the execution environment
MagicEtfSUBaseType	
magicEtfSutName	This attribute contains the name of the service unit base type
MagicEtfSUType	
magicEtfSutVersion	This attribute specifies the version for the service unit type
magicEtfSutSuFailOver	This attribute specifies whether AMF should fail over all CSIs of SIs for the SUs of the AMF types derived from this ETF type or not. In other words, the software implementation of components of the service unit is such that the failure of one component impacts the entire SU
MagicEtfSGBaseType	
magicEtfSgtName	This attribute specifies the name of the service group base type
MagicEtfSGType	
magicEtfSgtVersion	This attribute specifies the version for the service group type
magicEtfSgtAutoAdjustPeriod	This attribute specifies the recommended probation period for

	auto adjustment
magicEtfSgtAutoAdjustOption	This attribute specifies vendor's recommendation for the auto adjust option
magicEtfSgtRedundancyModel	This attribute specifies the redundancy model of the service group type
magicEtfSgtAutoRepairOption	This attribute contains a Boolean value that specifies whether AMF is permitted to initiate automatic repair actions within an SG or not
magicEtfSgtCompProbPeriod	This attribute contains the recommended probation time for the components inside a service group
magicEtfSgtCompProbCounterMax	This attribute contains the recommended maximum number of AMF attempts to restart the components inside a service group
magicEtfSgtSuProbPeriod	This attribute contains the recommended probation time for the service units inside a service group
magicEtfSgtSuProbCounterMax	This attribute contains the vendor's recommendation for MagicSaAmfSGType::magicSaAmfSgtDefSuRestartMax AMF attribute
MagicEtfAppBaseType	
magicEtfApptName	This attribute specifies the name of the application base type
MagicEtfAppType	
magicEtfApptVersion	This attribute specifies the version for the application type
MagicEtfSwBundle	
magicEtfSwbName	This attribute specifies the name of the software bundle
magicEtfSwbRemovalOnlineCmd	This attribute contains the online – as assumed by the vendor–REMOVAL CLI command string of this software bundle, which also includes the relative path command.

	It needs to be adjusted to the execution environment
magicEtfSwbRemovalOnlineArgs	This attribute contains arguments of the online-as assumed by the vendor- REMOVAL CLI command of this software bundle and needs to be adjusted to the execution environment
magicEtfSwbInstallationOnlineCmd	This attribute contains the online-as assumed by the vendor- INSTALATION CLI command string of this software bundle, which also includes the relative path command. It needs to be adjusted to the execution environment
magicEtfSwbInstallationOnlineArgs	This attribute contains arguments to the online -as assumed by the vendor-INSTALATION CLI command of this software bundle and needs to be adjusted to the execution environment
magicEtfSwbRemovalOfflineCmd	This attribute contains the offline-as assumed by the vendor- REMOVAL CLI command string of this software bundle, which also includes the relative path of the command. It needs to be adjusted to the execution environment
magicEtfSwbRemovalOfflineArgs	This attribute contains arguments of the offline-as assumed by the vendor- REMOVAL CLI command of this software bundle and needs to be adjusted to the execution environment
magicEtfSwbRemovalOfflineImpactScope	This attribute contains the minimum scope of disruption during the removal operation of this software bundle and needs to be adjusted to the particular system based on system features
magicEtfSwbInstallationOfflineCmd	This attribute contains the offline-as assumed by the vendor- INSTALATION CLI command string of this software bundle, which also includes the relative path of the command. It needs to be

	adjusted to the execution environment
magicEtfSwbInstallationOfflineArgs	This attribute contains arguments of the offline—as assumed by the vendor—INSTALLATION CLI command of this software bundle and needs to be adjusted to the execution environment
magicEtfSwbInstallationOfflineImpactScope	This attribute contains the minimum scope of disruption during the installation operation of this software bundle and needs to be adjusted to the particular system based on system features
MagicEtfUpgradeAwarenessAttributes	
magicEtfInitCallback	This attribute specifies the parameters of the initiate callback (for initiation of a new upgrade campaign) if recognized by the component type
magicEtfBackupCallback	This attribute specifies the parameters of the backup (to create an application level backup) callback if recognized by the component type
magicEtfRollbackCallback	This attribute specifies the parameters of the rollback (for rolling back the campaign) callback if recognized by the component type
magicEtfCommitCallback	This attribute specifies the parameters of the commit callback (to indicate the commitment of campaign) if recognized by the component type
magicEtfOtherCallback	This attribute specifies the parameters of any other callback if recognized by the component type
MagicEtfHealthcheck	
magicEtfHctKey	This attribute specifies the key for this health check type
magicEtfHctVariant	This attribute specifies the technique for invoking the health check

magicEtfHctMaxDuration	This attribute defines the restriction for the period during which AMF expects a response to the health check callback from a component of the AMF component types, derived from component type associated with this health check type
magicEtfHctPeriod	This attribute specifies the restriction for the time interval at which the health check is performed, which is used by health check entities of the AMF health check type derived from this type
MagicEtfSvcBaseType	
magicEtfSvctName	This attribute contains the name of the base service type
MagicEtfSvcType	
magicEtfSvctVersion	This attribute specifies the version for the service type
MagicEtfCSBaseType	
magicEtfCstName	This attribute contains the name of the component service base type
MagicEtfCSType	
magicEtfCstVersion	This attribute specifies the version for the CSType
MagicEtfCstAttribute	
magicEtfAttrName	This attribute contains the name of the CSI attribute which is specified in this class
magicEtfAttrType	This attribute contains the type of the CSI attribute which is specified in this class
magicEtfAttrUpperBound	This attribute contains the upper bound for the CSI attribute which is specified in this class
magicEtfAttrLowerBound	This attribute contains the lower

	bound for the CSI attribute which is specified in this class
magicEtfAttrDefault	This attribute contains the default value for the CSI attribute which is specified in this class
MagicEtfSvctCst	
magicEtfMinNumInstances	This attribute specifies the minimum number of component service instances of the AMF CTypes derived from the associated CType in a service instance of AMF service type derived from the associated service types
magicEtfMaxNumInstances	This attribute specifies the maximum number of component service instances of the AMF CTypes derived from the associated CType in a service instance of AMF service type derived from the associated service types
MagicEtfCtSut	
magicEtfMinNumInstances	This attribute specifies the minimum number of components of the AMF component type derived from the associated component type in a service unit of AMF service unit type derived from the associated service unit type
magicEtfMaxNumInstances	This attribute specifies the maximum number of component of the AMF component type derived from the associated component type in a service unit of AMF service unit type derived from the associated service unit types
MagicEtfCtCSType	
magicEtfDefaultNumStandbyCsi	This attribute defines the recommended default number of standby assignments for

	components of the AMF component type derived from the associated component type
magicEtfMaxNumStandbyCsi	This attribute describes the capability of software(the maximum what the implementation of software can handle) to act as standby. In other words it defines the maximum number of standby assignments
magicEtfDefaultNumActiveCsi	This attribute describes the recommended default number of active assignments for components of the AMF component type derived from the associated component type
magicEtfMaxNumActiveCsi	This attribute describes the capability of software(the maximum what the implementation of software can handle) to act as active. In other words it defines the maximum number of active assignments of the components
magicEtfCompCapabilityModel	This attribute defines the highest level of component capability model that the software implementation is capable of handling.

CR Sub-profile Tagged Definitions

Tagged Definition	Description
MagicCrAdministrativeDomain	
magicCrAdminDomainName	This attribute specifies the name of the administrative domain
MagicCrSgTemplate	
magicCrSgTempName	This attribute specifies the name of the SG template
magicCrSgTempRedundancyModel	This attribute specifies the redundancy model according to which we want the SG to protect the SIs.
magicCrSgTempNumberOfActiveSus	This attribute specifies the number of active SUs in the SG.
magicCrSgTempNumberOfStdbSus	This attribute specifies the number of standby SUs in the SG.
magicCrSgTempNumberOfSpareSus	This attribute specifies the number of spare SUs in the SG.
magicCrPropSgTempFactor	This attribute specifies the number of time a proportion is repeated.
MagicCrSiTemplate	
magicCrSiTempName	This attribute specifies the name of the SI template
magicCrSiTempNumberOfActiveAssignments	This attribute specifies the number of active assignment each SI of the SI template will acquire at runtime
magicCrSiTempNumberOfStdbAssignment	This attribute specifies the number of standby assignment each SI of the SI template will acquire at runtime
MagicCrRegularSiTemplate	
magicCrRegSiTempNumberOfSis	This attribute specifies the total number of SIs of the regular template

magicCrRegSiTempMinSis	This attribute specifies the minimum number of SIs of the regular SI template required in one SG
magicCrRegSiTempMaxSis	This attribute specifies the maximum number of SIs of the regular SI template allowed in one SG
MagicCrProportionalSiTemplate	
magicCrPropSiTempProportion	This attribute specifies the ratio in which the SIs of this template are required to be present in comparison to the SIs of other proportional SI templates
MagicCrCsiTemplate	
magicCrCsiTempName	This attribute specifies the name of the CSI template
magicCrCsiTempNumberofCsis	This attribute specifies the number of CSIs to be created based on the CSI template