A Model Driven Approach for the Generation of Configurations for Highly-Available Software Systems

PEJMAN SALEHI[1], ABDELWAHAB HAMOU-LHADJ[2], MARIA TOEROE[3], FERHAT KHENDEK[2]

[1]*School of Engineering and Information Technology, Conestoga College, Oakville, Canada*

[2]*Electrical and Computer Engineering Department, Concordia University, Montréal, Canada*

[3]*Ericsson Inc. Montréal, Canada*

psalehi@conestogac.on.ca (corresponding author)

abdelw@ece.concordia.ca

khendek@ece.concordia.ca

maria.toeroe@ericsson.com

Abstract

High availability of services is an important requirement for mission critical systems. The Service Availability Forum has defined standards to support the realization of high available systems. Among these standard, the Availability Management Framework (AMF) is perhaps the most important one. AMF is a middleware service that coordinates redundant application components to ensure the high availability of the services. AMF requires a configuration that describes the provided services, their types, and the deployment infrastructure. The process of generating an AMF configuration takes as input the description of the software characteristics as well as the configuration requirements that specify the services to be provided. Due to the large number of parameters to be taken into account, the generation of an AMF configuration can be a difficult and error prone task. This paper proposes a new approach for the automatic generation of AMF configurations. The proposed solution is model driven and is based on UML profiles which capture the concepts related to configuration requirements, software description, and AMF configurations. AMF configurations are generated using ATL based transformations defined between these different profiles.

Keywords: High-availability, software dependability, model-driven software configuration

1 Introduction

High availability of a software system is achieved when the services are available to users 99.999% of the time [Piedad 2001]. The demand for highly available services is continuously growing in different domains, such as banking, health care, telecommunication, air traffic monitoring, and so on. Service outage in such systems may lead to important financial losses or life injuries.

The Service Availability Forum (SA Forum) [SAF 2015] is a consortium of telecommunications and computing companies that has defined several standard interfaces to support the development of Highly Available (HA) systems. These standards aim at reducing the time and cost of developing HA applications by shifting availability management from applications to a dedicated middleware, hence enabling portability among platforms and utilizing Commercial-Off-The-Shelf (COTS) building blocks for the development of HA systems. Among the SA Forum standards, the Application Interface Specification (AIS) [SAF 2010a] supports the development of HA applications by abstracting hardware and software resources. There are several implementations of AIS provided by different groups among which the most popular implementation is called OpenSAF and is currently being supported by various telecommunications and software companies through OpenSAF Foundation [OpenSAF 2015].

The main service offered by AIS is the Availability Management Framework (AMF) [SAF 2010b], which manages the high availability of applications through the use of redundancy models. In order to provide and protect an application's services, AMF requires a configuration that specifies the characteristics of the entities and their organization. These entities describe the service providers, the provided services, their types, and the deployment infrastructure. The management of AMF configurations consists of different activities, namely design, validation, analysis and upgrade from time to time. These activities require proper tool support. The goal of the MAGIC[1] project is the definition of a model driven framework for the design, validation and upgrade of AMF configurations. This framework required a modeling language to support the specification and validation of AMF configurations, which we defined in previous work as UML profiles [Salehi 2015]. One key application of the framework is the automated design of AMF

[1] MAGIC (Modeling and Automatic Generation of Information and upgrade Campaigns for Service Availability) project is a joint project between Concordia University and Ericsson Software Research. http://encs.concordia.ca/~magic/

configurations as it is a challenging task if done manually [SAF 2010c]. This is because of the large number of entities that are involved and the complexity of the relationships between these entities. This design consists of generating AMF configurations from 1) the descriptions of the software resources to be used and the description of the deployment infrastructure, and 2) the requirements that specify the AMF services to be provided.

In this paper, we present a model based approach for generating AMF configurations using the UML profile that we developed in previous work [Salehi 2015]. Our approach consists of a set of transformation rules expressed in a declarative style and defined among different elements of our profiles. AMF configurations are generated through the application of the transformation rules to the model elements representing software entities and configuration requirements. These rules abstract from the operational steps to be performed in order to generate the target elements. Transformation rules are illustrated using the ATLAS Transformation Language (ATL) [Jouault 2008].

The rest of the paper is organized as follows. Section 2 provides the overview of the related work. Section 3 introduces the modeling framework composed of the UML profiles used to enable the model based generation of AMF configurations. In Section 4, we describe the model driven approach for generating AMF configurations. Section 5 presents the implementation of the approach through a case study, followed by the discussion regarding the validation of our approach in Section ~~5~~6. [WH1]Finally, we conclude in Section 7 and discuss the future works in Section 6.

2 Related Work

Kanso et al. [Kanso 2008 and Kanso 2009] proposed pure, code centric, algorithmic and imperative approaches to generate AMF configurations. The proposed approaches are specified at a level of abstraction which is not flexible to domain model changes and often small changes resulted in large modifications to the code. The model driven approach presented in this paper overcomes these shortcomings while taking full advantage of the benefits that come with a full-fledged model based approach. In the work of Turenne et al. [Turenne 2014a and Turenne 2014b], the authors improve upon the work of Kanso et al. [Kanso 2008 and Kanso 2009] by adding a tool to generate a model for the description of software components. This description is

provided by the software vendor in the form of another SA Forum standard, known as the Entity Types File (ETF) [SAF 2010c]. In other words, their proposed approach generates the description of the software components which will be used in the code centric approach introduced in their earlier works.

Buskens et al. [Buskens 2006] presented an HA middleware, called the Aurora Management Workbench (AMW), as well as a set of tools for building highly available software systems in a model centric way. In their approach, the HA related code is generated as part of the software components' code and therefore, the configuration of the system remains the same for the entire life cycle of the software system. Consequently, configuration generation is not part of their approach, since the designed software runs based on a fixed configuration. They adopt this approach in order to avoid the complexity of standardized APIs. Our approach however, focuses on generating configurations for a system composed of SA Forum standard compliant software components, rather than generating the code for software components.

Szatmári et al. and Kovi et al. [Szatmári 2008 and Kovi 2007] introduced an MDA (Model-Driven Architecture) approach for the automatic generation of SA Forum compliant applications. They also introduced a metamodel based on AMF specification. Based on the authors' approach, an application is first modeled using their metamodel (Platform Independent Model) and then mapped to the APIs (Platform Specific Model) which represent the implementation of SA Forum services. This work, however, concentrates more on the development of the software components rather than on configuration generation.

Within the field of software management, there exist studies on configuration generation, particularly in the case of work involving constraint satisfaction techniques and policies [Hinrich 2004 and Sahai 2004]. For instance, Sahai et al. [Sahai 2004] present an approach for generating a configuration based on a set of user requirements, operators, and technical constraints. Recognizing these constraints, their method of generating a configuration is formulated as a resource composition problem. Although some features of their work overlap with our method, these two approaches differ significantly. While the work of Sahai et al. [Sahai 2004] focuses on general utility computing environments, our work concentrates on availability and the AMF domain model. Moreover, Sahai et al. base their approach on constraint satisfaction techniques, whereas our work focuses on model transformation.

3 Background

3.1 The Modeling Framework

The modeling framework covers three different domains: The AMF domain [SAF 2010b], Entity Type Files (ETF) [SAF 2010c], and configuration requirements. The rest of this section describes these domains and introduces the UML profiles for them. These profiles contain the complete and comprehensive definitions of all the domain concepts, their relationships, as well as the constraints that exist among them.

3.1.1 AMF Profile

An AMF configuration for a given application is a logical organization of resources for providing and protecting services. The AMF configuration profile models both the resources and the services.

In an AMF configuration, resources and services are represented through a logical structure that allows AMF to manage resources in order to provide service availability. An AMF configuration consists of two different set of concepts: AMF entities and AMF entity types. AMF entities are categorized into different logical entities representing services and software/hardware service provider's resources. All these logical entities are modeled in terms of UML stereotypes. The basic entities are called Components. Components represent HW and/or SW resources capable of supporting the workload imposed by service functionalities. Such a workload is referred to as Component Service Instance (CSI). Components are aggregated into Service Units (SU), logical entities representing the basic redundancy unit for AMF.

The aggregation of components favors the combination of their functionalities into higher level services. More specifically, the workloads of the components contained in an SU are aggregated into a Service Instance (SI), which represents the workload assigned to the SU. SUs are further grouped into Service Groups (SG) to protect a set of SIs by means of redundancy. SGs are classified according to the redundancy models used by AMF to protect the services. AMF supports the 'No redundancy', 2N, N+M, N-Way and N-Way-Active redundancy models. Finally, an AMF application combines different SGs in order to provide the SIs protected by these SGs. Each SU is deployed on an AMF node and the set of all AMF nodes forms the AMF cluster.

AMF entity types define the common characteristics among multiple instances of the previously defined logical entities. In AMF, all entities except the deployment entities (i.e, node and cluster) have a type.

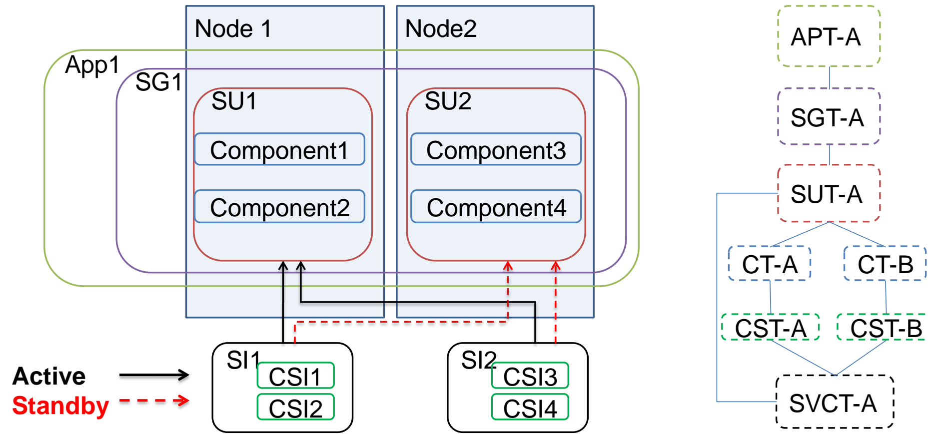

Figure 1 An example of AMF configuration

Figure 1 shows an example of an AMF configuration. In this example, a cluster is composed of two nodes (Node1 and Node2). It hosts an application consisting of one service group protecting two service instances in a 2N redundancy model. The service group consists of two service units, SU1 and SU2, each composed of two components. The distribution of the active and standby assignments is shown in this figure. However it is not part of the configuration as defined by AMF as this is decided by AMF at runtime. The entities presented in the configuration are described by means of the following AMF types: Component1 and Component3 are from the ComponentType CT-A, while Component2 and Component 4 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.

In [Salehi 2015], we build AMF profile, a domain specific modeling language (DSML) tailored to AMF domain concepts, semantics, and syntax. AMF profile was built by extending the Unified Modeling Language (UML). This profile is mainly designed to support the design, specification, analysis of AMF configurations. In addition to that, we have also used the AMF profile for validation of the AMF configurations [Salehi 2009, Salehi 2011].

3.1.2 The Entity Type Files (ETF) Profile

Vendors provide a description of their software in terms of implemented entity types by means of XML files called ETF [SAF 2010c]. These ETF types specify intrinsic characteristics of the software as well as its capabilities and limitations. Moreover, they describe how the software entities can be combined by providing information regarding their dependencies and compatibility options.

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. As a result, the UML profile that models the ETF types has a structure similar to the AMF Entity Type package.

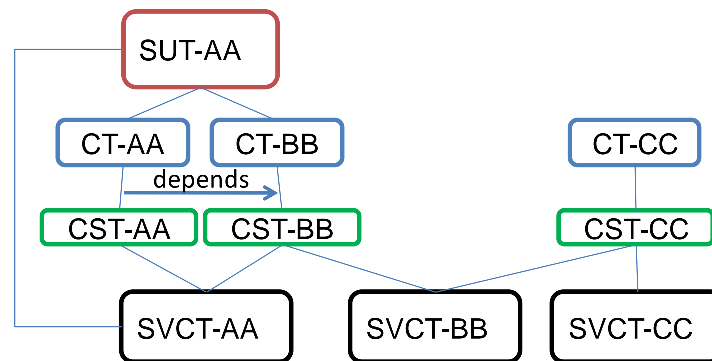

Figure 2 An example of ETF model

An ETF model 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 limitations and constraints of the application. However, they do not have to be provided in ETF.

For instance, Figure 2 describes the ETF types that have been used to generate the AMF configuration shown in Figure 1. 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 which provides SVCT-AA.

The ETF profile was introduced in [Salehi 2014] and similar to the AMF profile was designed as an extension to the UML metamodel. This profile mainly focuses on the description of the software system going to be deployed on AMF middleware and therefore, is used in the process of AMF configuration design.

3.1.3 The Configuration Requirements (CR) Profile

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 specification of the configuration requirements is defined as templates to help the configuration designer specify common characteristics shared by multiple SIs (using SITemplates) and CSIs (by means of CSITemplates). The CSITemplate defines the CSType with the required number of CSIs. The SITemplate specifies the SIs with the associated SvcType, the required redundancy model to protect them, and the associated CSITemplates.

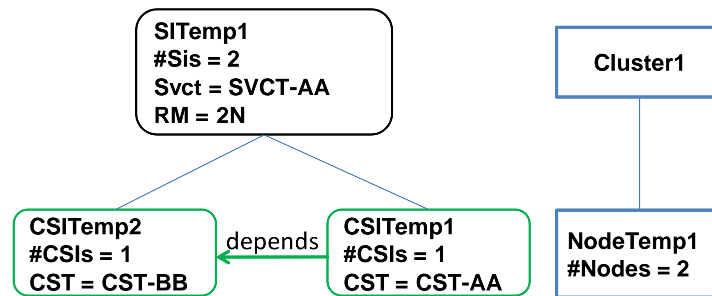

Figure 3 An example of CR model

Figure 3 shows an example of configuration requirements model for which the configuration in Figure 1 is generated from the ETF model in Figure 2. This configuration requirement model specifies an SITemplate named SITemp1 that aggregates two CSTemplates called CSITemp1 and CSITemp2. SITemp1 specifies the requirements for two SIs of type SVCT-AA and the protection level of 2N redundancy model. CSITemp1 and CSITemp2 require one CSI from the type CST-AA and CST-BB, respectively. The model also specifies a dependency requirement that inquires the services which will be created based on CSITemp1 need to depend on the

services based on CSITemp2. Finally, the required deployment infrastructure is specified in terms of NodeTemplate and the properties of the cluster are modeled using an element called Cluster.

4 A Model-Driven Approach for AMF Configuration Generation

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.

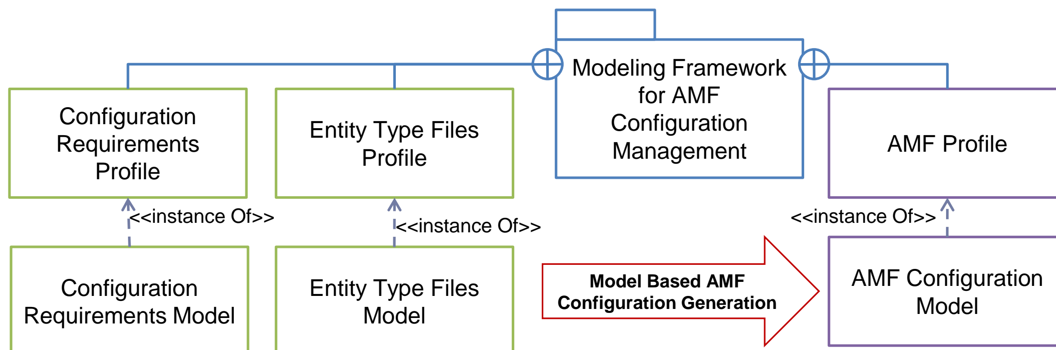

Figure 4 The overall process of model-based AMF configuration generation

Figure 4 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.

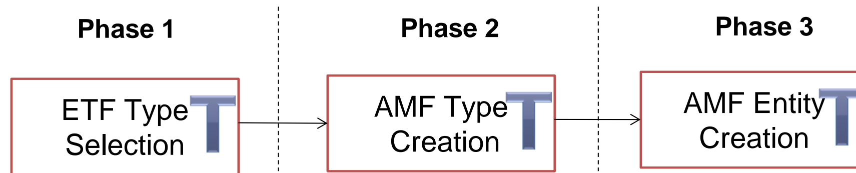

Figure 5 The main phases of the model transformation approach

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 paper 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.

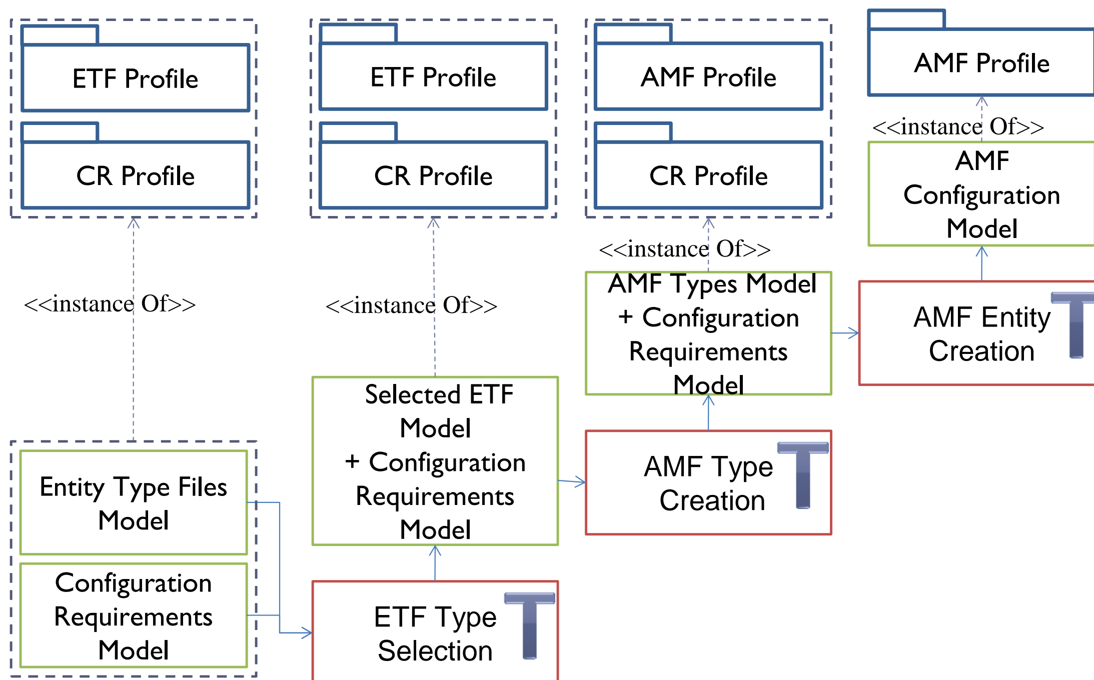

Figure 6 The relation between the models and the transformation phases

As shown in Figure 5, 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 illustrates the input and output models and their referenced metamodels.

4.1 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.

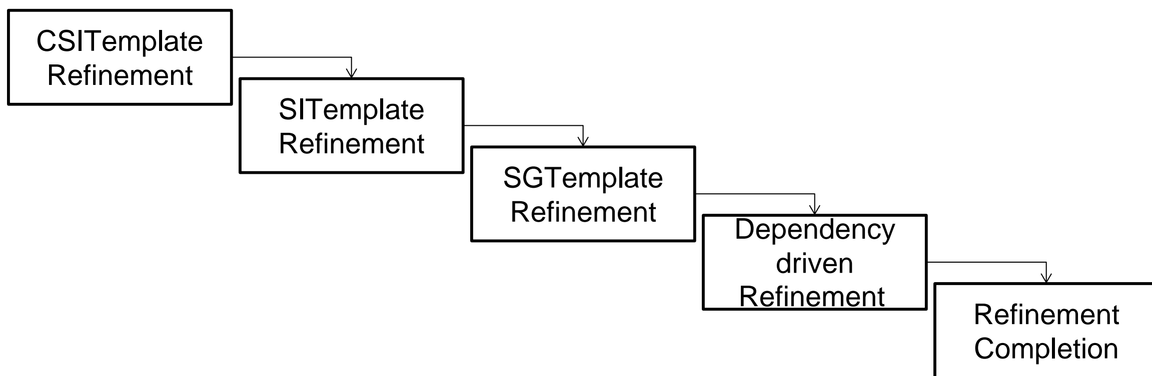

Figure 7 The transformation steps for ETF Type Selection phase

As shown in Figure 7, 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 forth 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 8 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.

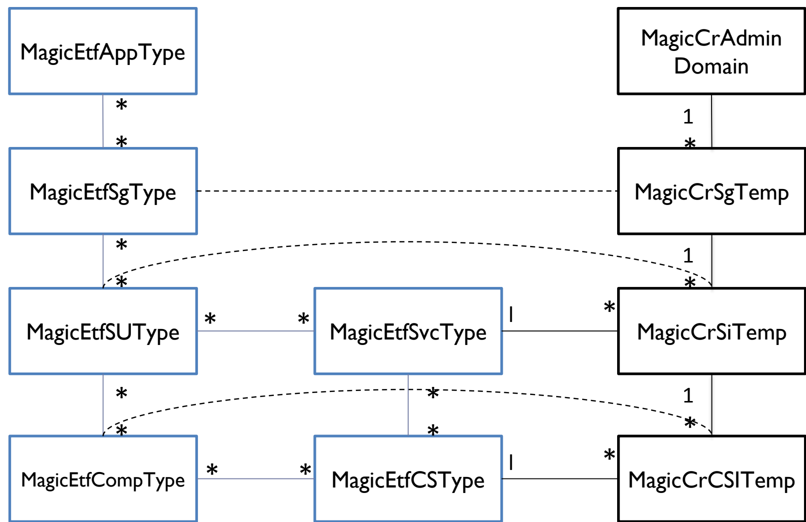

Figure 8 The result of the ETF Type Selection from the metamodel perspective

4.1.1 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. 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.
2. The compliance of the Component Type capability model (with respect to the CSType) with the redundancy model specified by the parent SGTemplate. 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.
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. 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. Besides, for each Component Type, the ETF specifies also the maximum number of CSIs active/standby assignments of each supported CSType. Therefore, 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.

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). 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 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 9 illustrates the refinement process using a UML activity diagram. This figure represents the control flow which regulates the usage of each selection criterion.

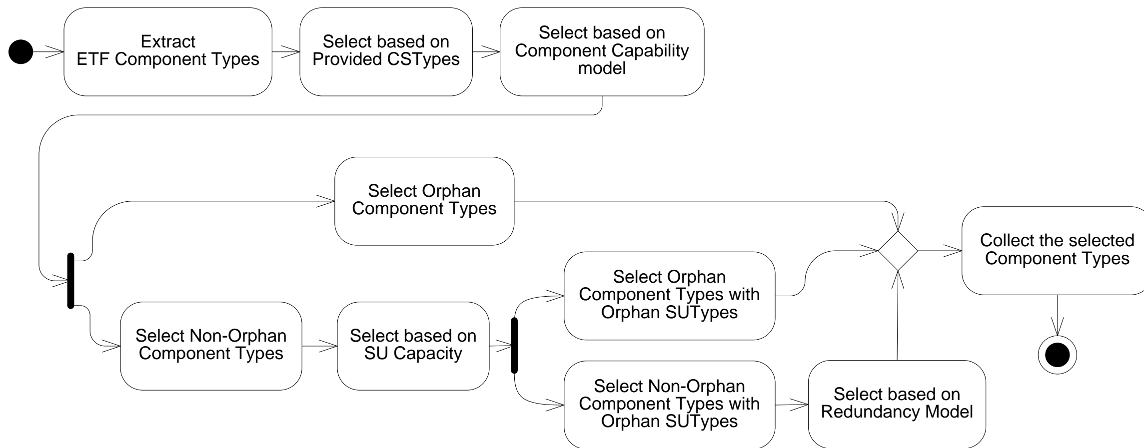

Figure 9 The activity diagram describing the selection of ETF Component Types

The component type selection requires visiting both input models (see Figure 6), 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 9). 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.

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.

4.1.2 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. 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.
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). 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.
3. The existence of links (resulting from the CSITemp refinement) between Component Types of the SUType and CSITemplates of the SITemplate. 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.

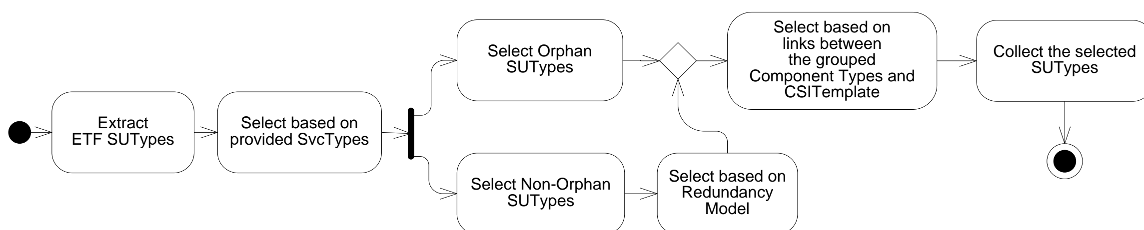

Figure 10 The activity diagram describing the selection of ETF SUTypes

The UML activity diagram in Figure 10 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.

4.1.3 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 SGType with the required redundancy model in SGTemplate. 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.
2. At least one SUType of the SGType has to provide all the SvcTypes associated with the SITemplates grouped in the SGTemplate. 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.

The UML activity diagram in Figure 11 represents the process to select SGTypes base on these mentioned criteria.

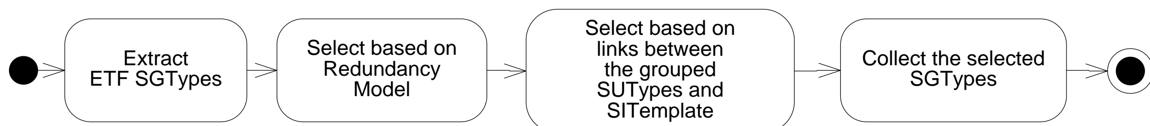

Figure 11 The activity diagram describing the process of selecting ETF SGTypes

```
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 11.

4.1.4 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.

4.1.5 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.

4.2 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, this transformation phase takes as input the ETF model refined by the previous transformation phase described in 4.1. 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.

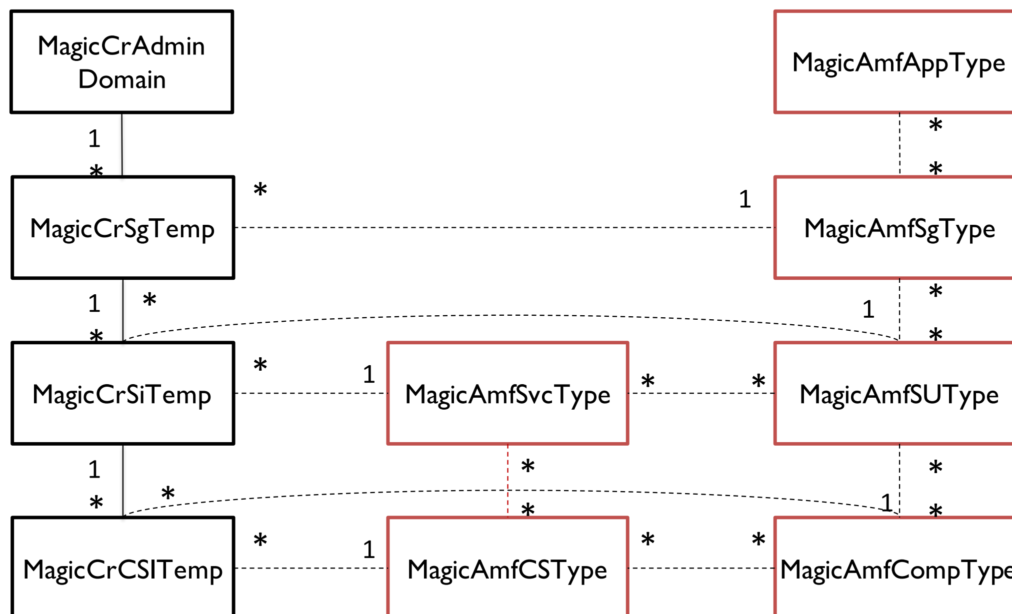

Figure 12 The result of the AMF Entity Type creation phase from the metamodel perspective

Figure 12 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.

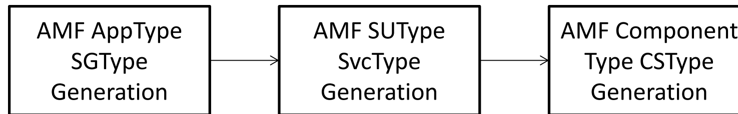

Figure 13 The transformation steps of the AMF entity type creation phase

Figure 13 presents he AMF entity type creation phase as composed of four different steps. 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 CSTypes. Therefore, SUTypes, SGTypes, AppTypes and SvcTypes might not exist in the ETF model. The refinement phase described in the previous section do es 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.

4.2.1 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.

4.2.2 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.

4.2.3 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 4.2.2).
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 4.2.2).

4.3 AMF Entity Creation

As shown in Figure 6, 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).

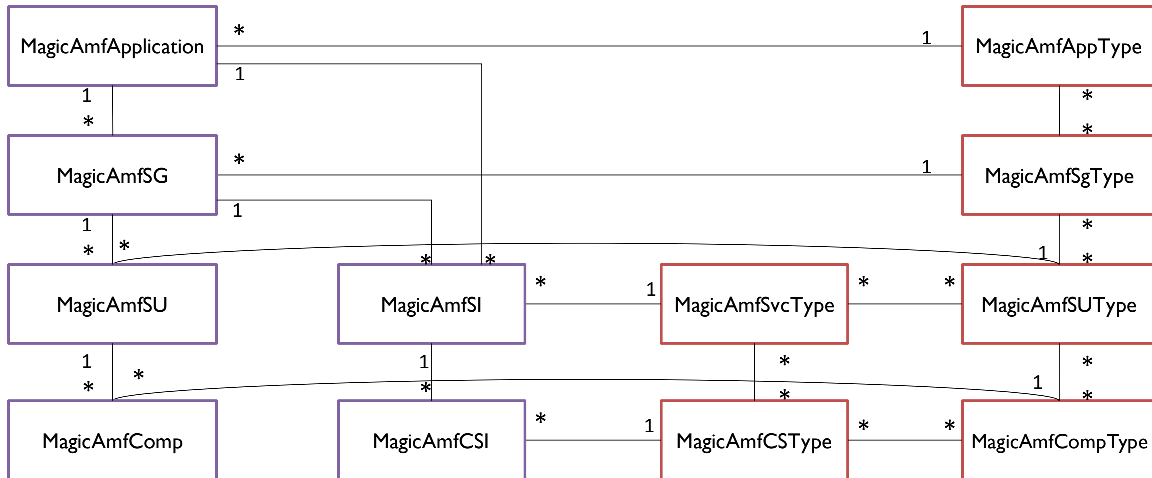

Figure 14 The result of the AMF Entity creation from the metamodel perspective

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 14 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.

4.3.1 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.

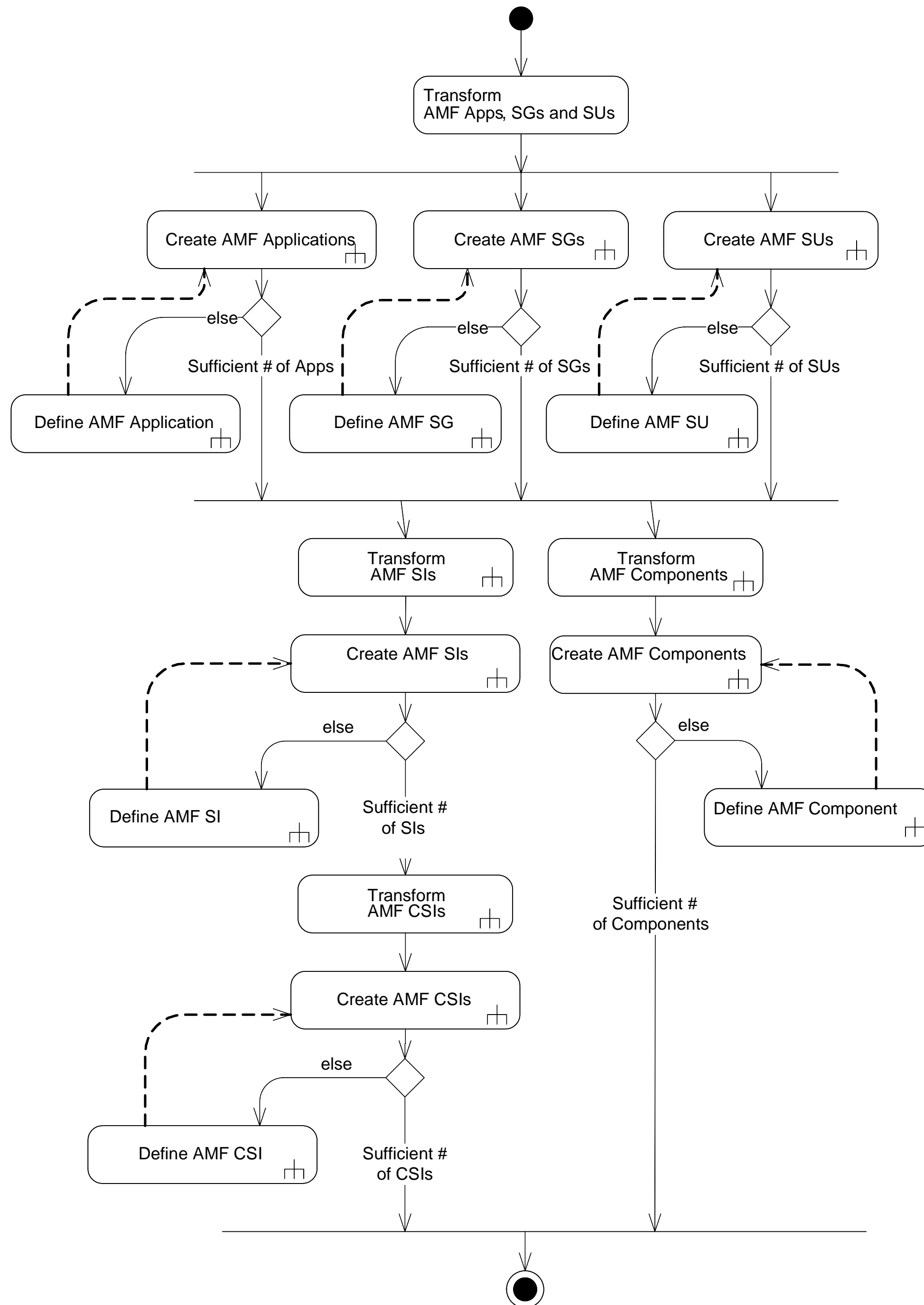

Figure 15 The flow of transformations to generate AMF entities

Figure 15 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.

```
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.

4.3.2 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.

4.3.3 Step 3: Finalizing the Generated AMF Configuration

As previously presented in Figure 6, 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.

5 Implementation of the Approach and Case Study

To demonstrate the effectiveness of AMF configuration generation approach, we used our model-based approach to develop a configuration for an online banking system which allows customers to conduct financial transactions using a secure web interface. In this section, 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.

5.1 Implementation of the Model-based Configuration Generation Tool

We implemented the process for generating model-driven configuration using ATLAS Transformation Language (ATL). ATL [Jouault 2008], a model transformation language, constitutes part of the Atlas Model Management Architecture (AMMA) platform and was created in response to the OMG MOF2.O /QVT RFP [OMG 2011]. ATL is used in the transformation scheme shown in Figure 16, permitting the transformation of the source model Ms, an instance of the source metamodel MMs, into the target model Mt, an instance of the target metamodel MMt.

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 2015a], a sub-project of the Eclipse Modeling Project [Eclipse 2015b]. 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.

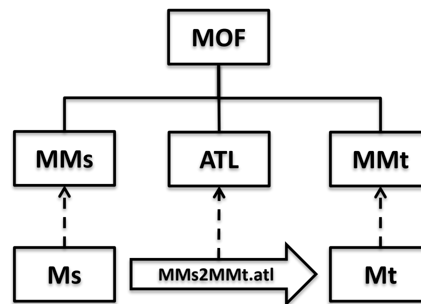

Figure 16 ATL Transformation scheme

5.2 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.

5.2.1 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 17 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 CSTypes). ViewBill depends on the EPostCommunication Component Type and PayBill is sponsored by ExtenalAccountManager through its ExternalBankCommunication CSType.

5.2.2 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 18).

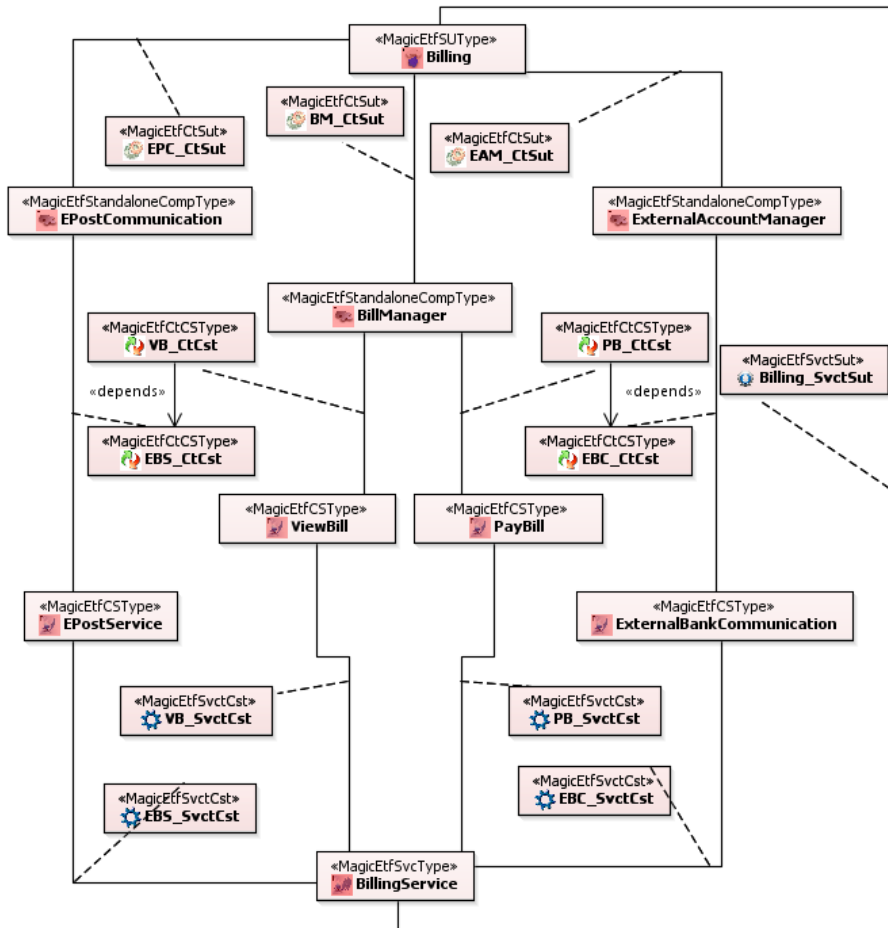

Figure 17 ETF model for billing part of an online banking software bundle

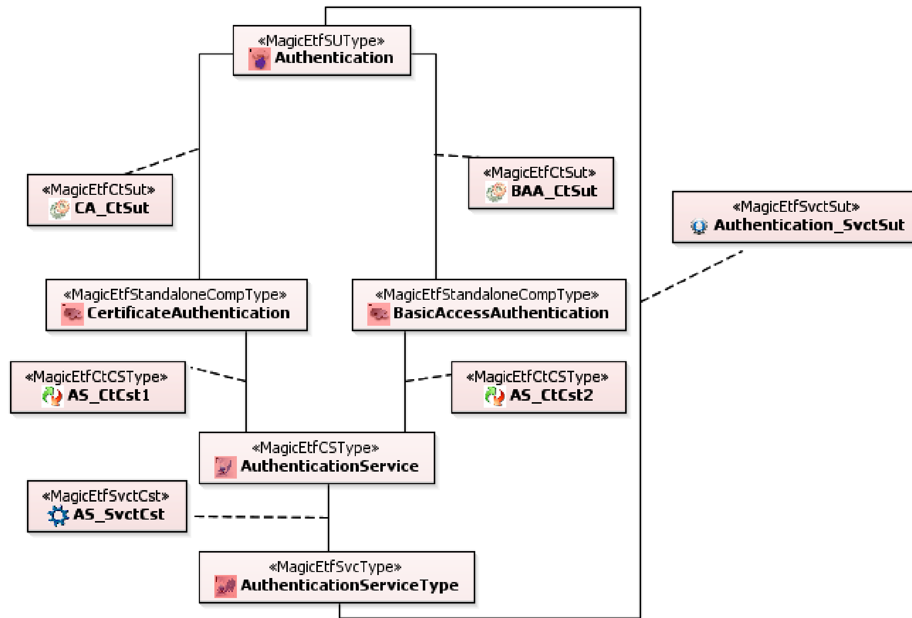

Figure 18 ETF model for the authentication part of an online banking software bundle

5.2.3 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 19):

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.

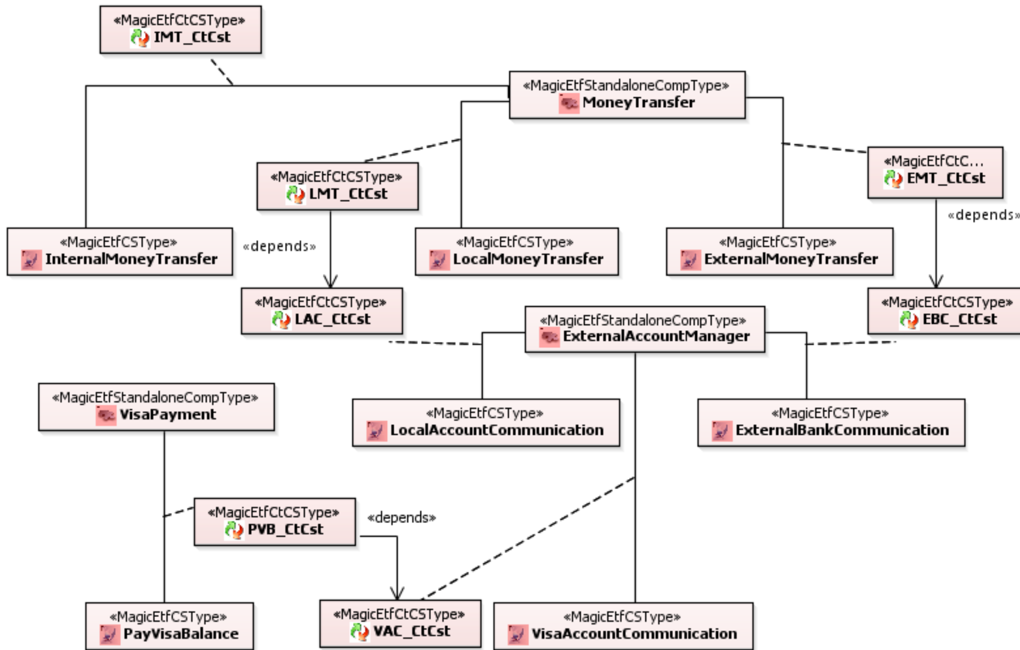

Figure 19 ETF model for money transfer part of online banking software bundle

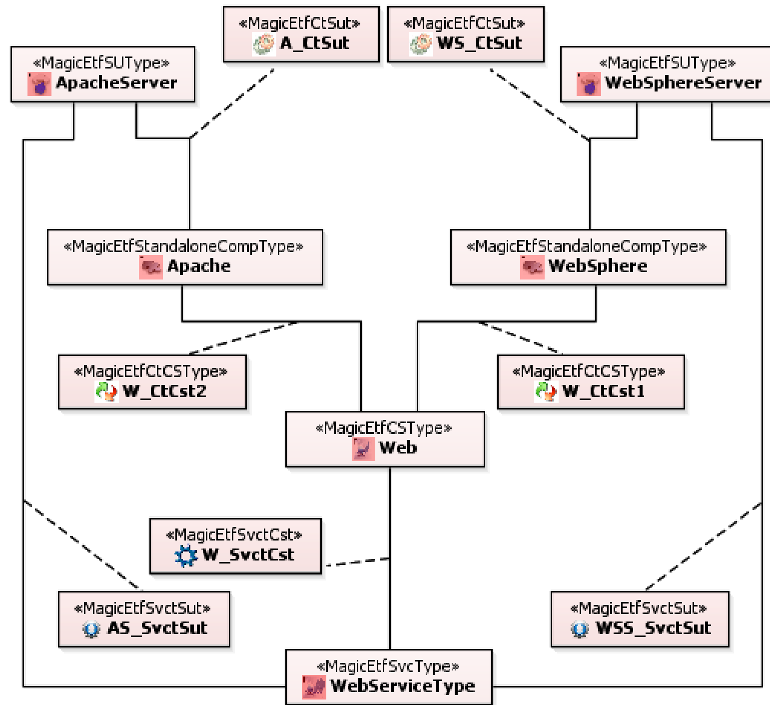

Figure 20 ETF model for web server part of online banking software bundle

5.2.4 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 20. 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).

The web based user interface of the online banking system consists of a set of web modules. In the ETF model in Figure 21 these web modules are presented in terms of ETF Component Types grouped into an SUType called UserInterface.

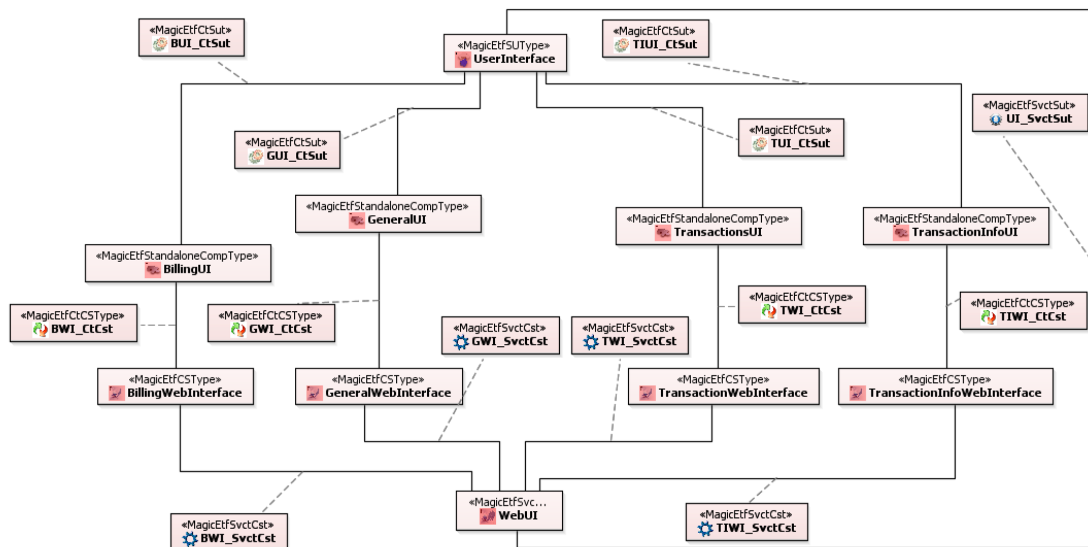

Figure 21 ETF model for user interface part of online banking software bundle

5.2.5 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 22).

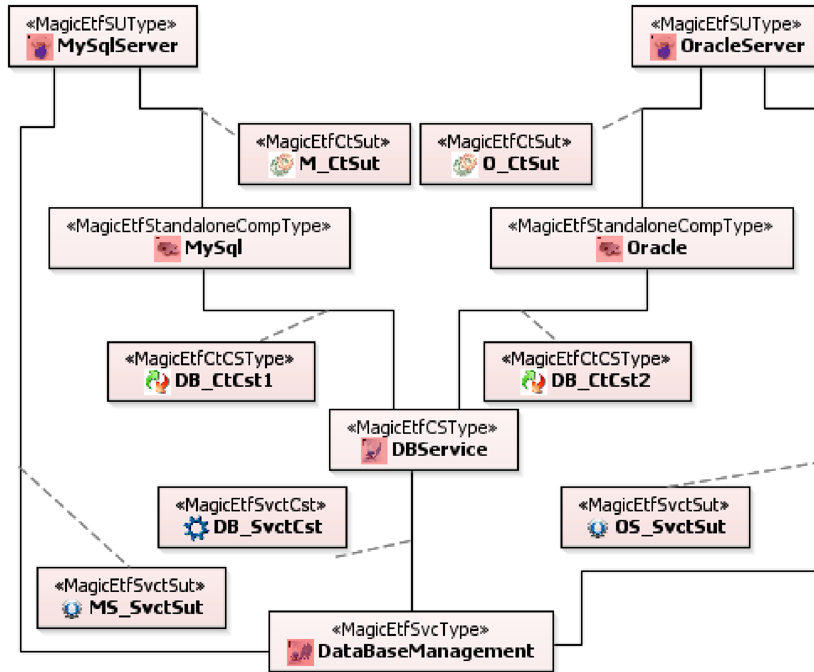

Figure 22 ETF model DBMS part of online banking software bundle

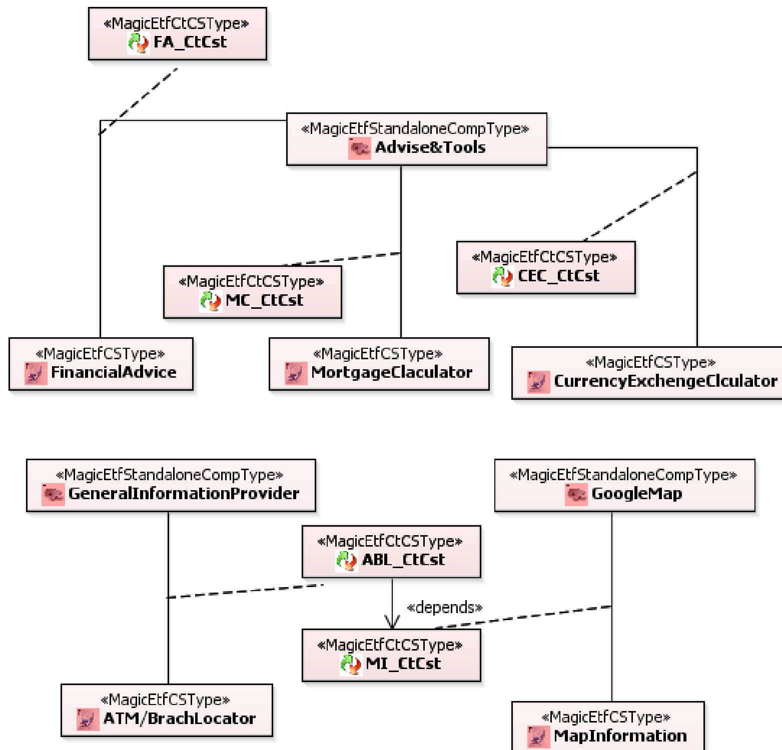

Figure 23 ETF model for the general inquiries part of an online banking software bundle

5.2.6 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 23 represents the ETF model describing the software entities for general inquiries. Advice&Tools Component Type provides FinancialAdvice, MortgageCalculator, and CurrencyExchangeCalculator CSTypes. General Information Provider Component Type provides the ATM/BranchLocator CSType sponsored by the MapInformation CSType which is provided by the GoogleMap Component Type.

5.2.7 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 24.

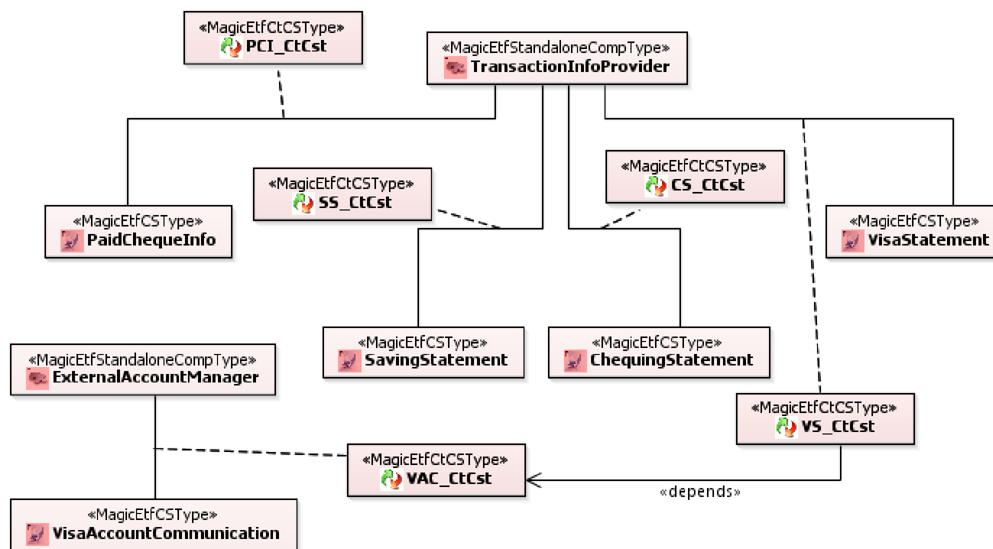

Figure 24 ETF model for the transaction information part of an online banking software bundle

5.2.8 SUType Level Dependency

The dependency between SUTypes of an online banking system is shown in Figure 25. 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.

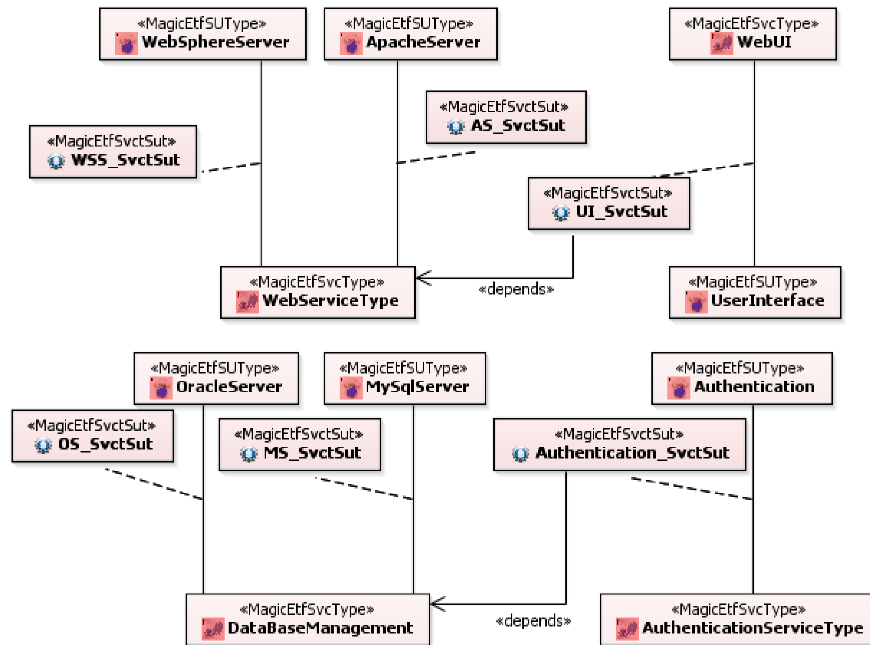

Figure 25 SUType level dependency

5.3 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 paper. 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 5.2, we generate an AMF configuration for the Safe Bank online banking system which satisfies these requirements.

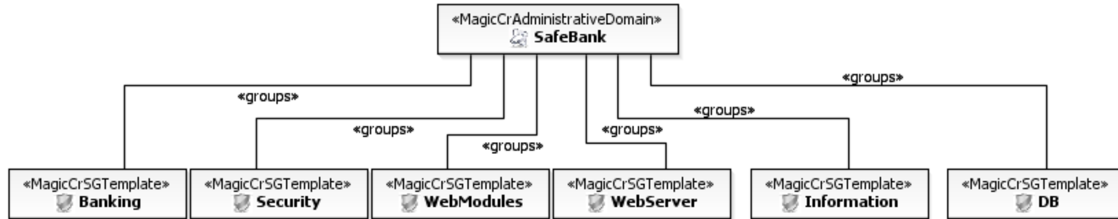

Figure 26 The SGTemplates of the Safe Bank online banking system

Figure 26 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 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 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 27). Table 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 section and are based on the parameters specified in the CR model.

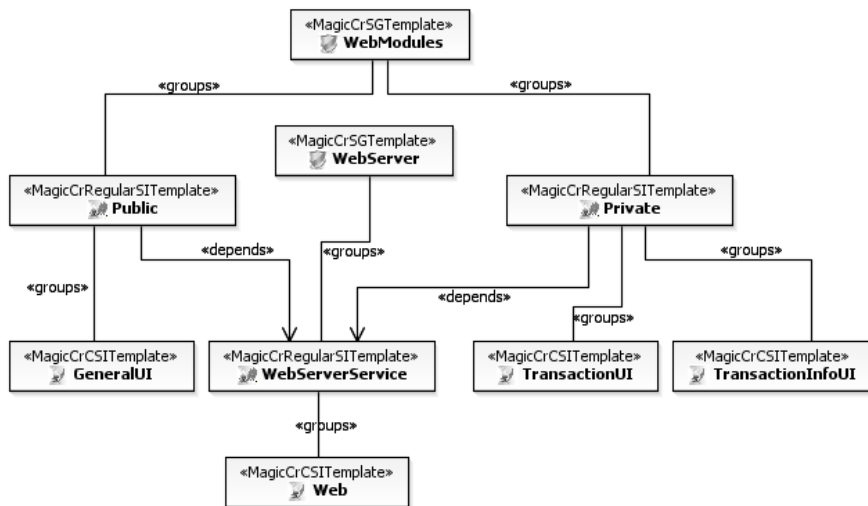

Figure 27 Configuration requirement elements of WebModules and WebServer SGTemplates

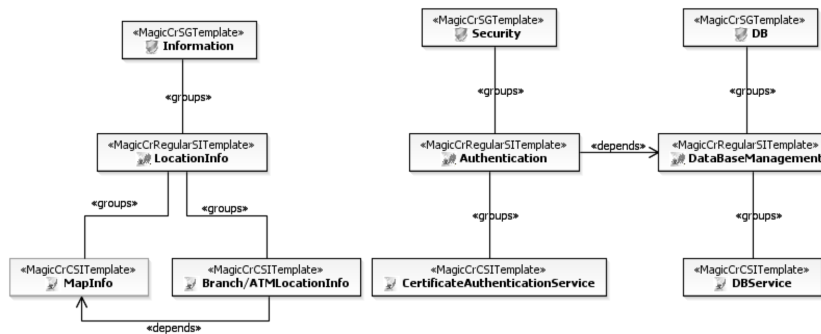

Figure 28 Configuration requirement elements of Security, Information, and DB SGTemplates

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

Attribute / **SITemplate**	Public	Private		WebServerService
magicCrSiTempSvcType	WebUI	WebUI		WebServiceType
magicCrSiTempNumberofActiveAssignments	1	1		1
magicCrSiTempNumberofStdbAssignment	1	1		1
magicCrRegSiTempNumberofSis	20	10		5
magicCrRegSiTempMinSis	10	10		5
expectedSIsperSG(Calculated)	*10*	*5*		*5*
activeLoadperSU(Calculated)	*10*	*5*		*2*
stdbLoadperSU(Calculated)	*10*	*10*		*5*
Attribute / **CSITemplate**	GeneralUI	TransactionUI	TransactionInfoUI	Web
magicCrCsiTempCsType	GeneralWebInfo	Transaction WebInterface	TransactionInfoWeb Interface	Web
magicCrCsiTempNumberofCsis	1	1	1	1

The configuration requirement elements defined for Security, Information, and DB SGTemplates are illustrated in Figure 28 and the values of their attributes are specified in Table 3.

Table 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	GeneralInqueries		DatabaseManagement
magicCrSiTempNumberofActiveAssignments	2	1		2
magicCrSiTempNumberofStdbAssignment	1	1		1
magicCrRegSiTempNumberofSis	5	1		5
magicCrRegSiTempMinSis	5	1		5
expectedSIsperSG(Calculated)	*5*	*1*		*5*
activeLoadperSU(Calculated)	*5*	*1*		*5*
stdbLoadperSU(Calculated)	*3*	*1*		*3*
Attribute / **CSITemplate**	CertificateAuthentication Service	Branch/ATM LocationInfo	MapInfo	DBService
magicCrCsiTempCsType	AuthenticationService	ATM/ BranchLocator	MapInfo rmation	DBService
magicCrCsiTempNumberofCsis	1	1	1	1

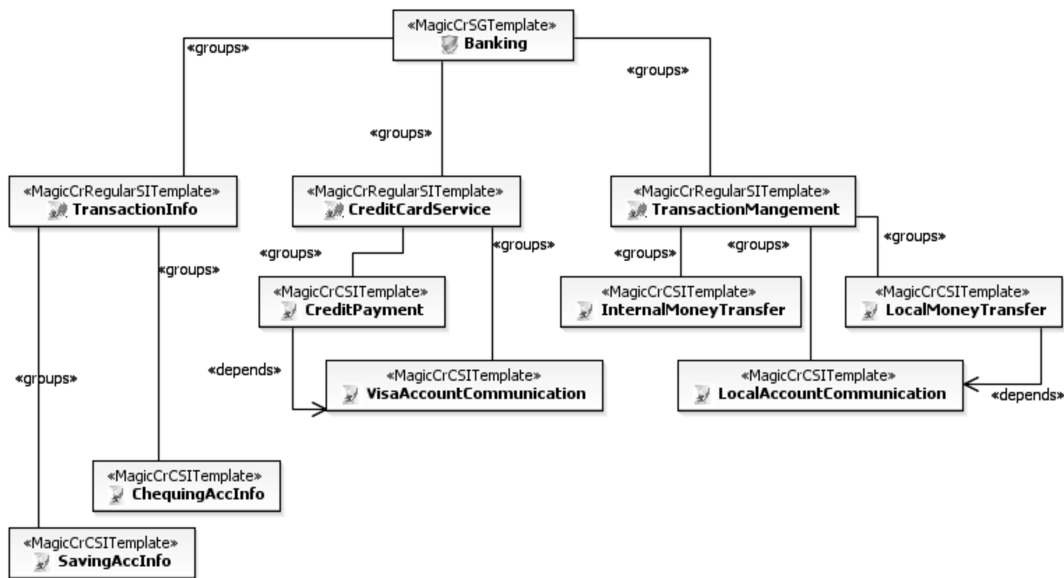

Figure 29 Configuration requirement elements of Banking SGTemplate

The configuration requirement elements defined for Banking SGTemplate are illustrated in Figure 29 and the values of their attributes are specified in Figure 29. 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 4 List of the values of attributes of SITemplates and CSITmplates of Banking SGTemplates

Attribute / **SITemplate**	Transaction Management
magicCrSiTempSvcType	TransactionService
magicCrSiTempNumberofActiveAssignments	1
magicCrSiTempNumberofStdbAssignment	1
magicCrRegSiTempNumberofSis	1
magicCrRegSiTempMinSis	1
expectedSIsperSG(Calculated)	*1*
activeLoadperSU(Calculated)	*1*

stdbLoadperSU(Calculated)	*1*		
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	
expectedSIsperSG(Calculated)	*1*		*2*	
activeLoadperSU(Calculated)	*1*		*1*	
stdbLoadperSU(Calculated)	*1*		*2*	
Attribute / **CSITemplate**	Credit Payment	VisaAccount Communication	Saving AccInfo	Chequing AccInfo
magicCrCsiTempCsType	PayVisaBalance	VisaAccount Communication	Saving Statement	Chequing 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 30 shows the CR elements for deployment infrastructure.

Figure 30 Configuration requirements for deployment infrastructure

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

5.4.1 Selecting ETF Types

The selection of ETF types is performed based on the rules in the steps presented in Section 4.1 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 31). 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.

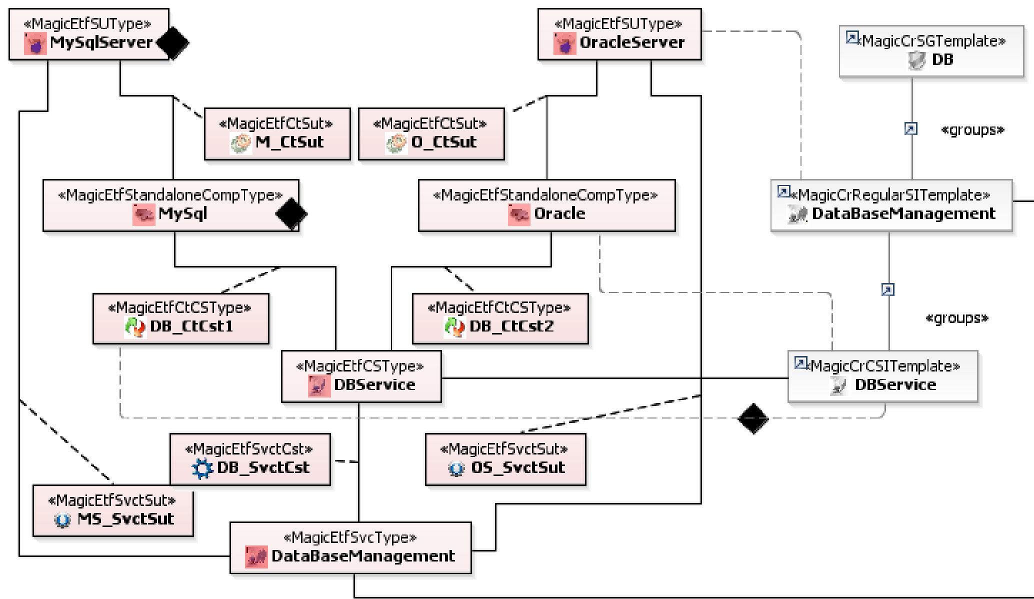

Figure 31 ETF Type selection phase for the DBMS part of online banking ETF

Since OracleServer provides the required SvcType and supports the required load, OracleServer SUType is selected for DatabaseManagement SITemplate in the SITemplate refinement step. Figure 31 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 32 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 CSTypes 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 32). Therefore, the selected ETF types successfully pass refinement step based on SI dependency presented in Section 4.1.4.

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.

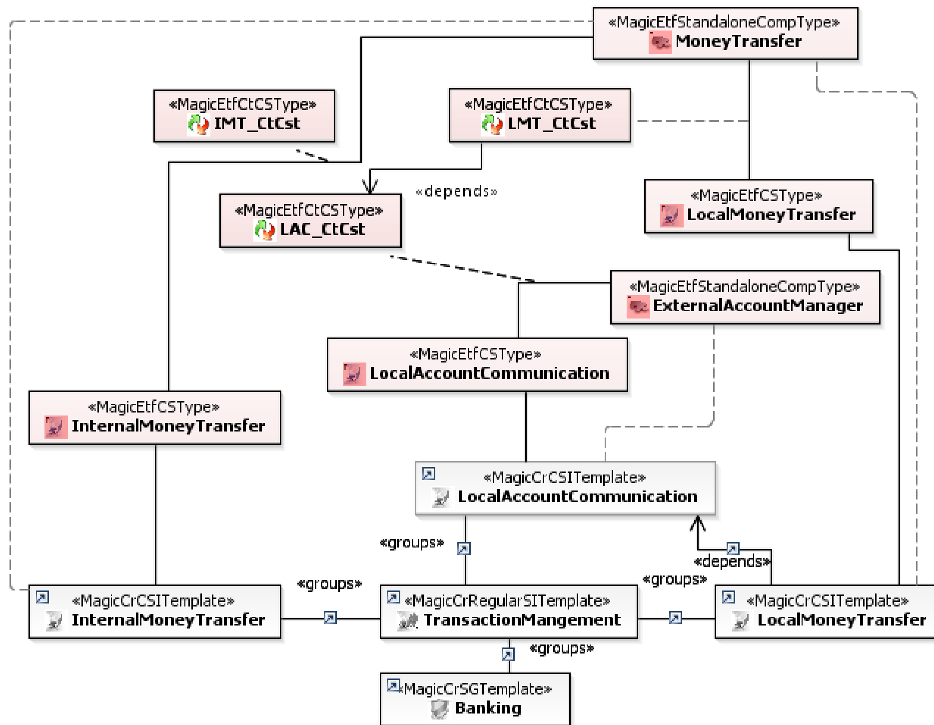

Figure 32 ETF Type selection phase for TransactionManagement SITemplate

5.4.2 Creating AMF Types

The next step is to create AMF types based on the selected the ETF types, For instance, Figure 33 shows the AMF types which were created based on the set of selected ETF types presented in Figure 31 of the previous section. This model is the result of applying the transformation steps of the AMF type creation phase (see Section 4.2) on the set of selected ETF types. More specifically, the AMF SGType called DB is created from scratch for DB SGTemplate, since there is no ETF SGType selected for this SGTemplate. Moreover, DataBaseManagement SITemplate, OracleServer AMF SUType and DataBaseManagement AMF SvcType are created based on OracleServer ETF SUType and DataBaseManagement ETF SvcType, accordingly. Finally, Oracle AMF Component Type and DBService AMF CSType are created based on Oracle ETF Component Type and DBService ETF CSType, respectively, and are linked to DBService CSITemplate.

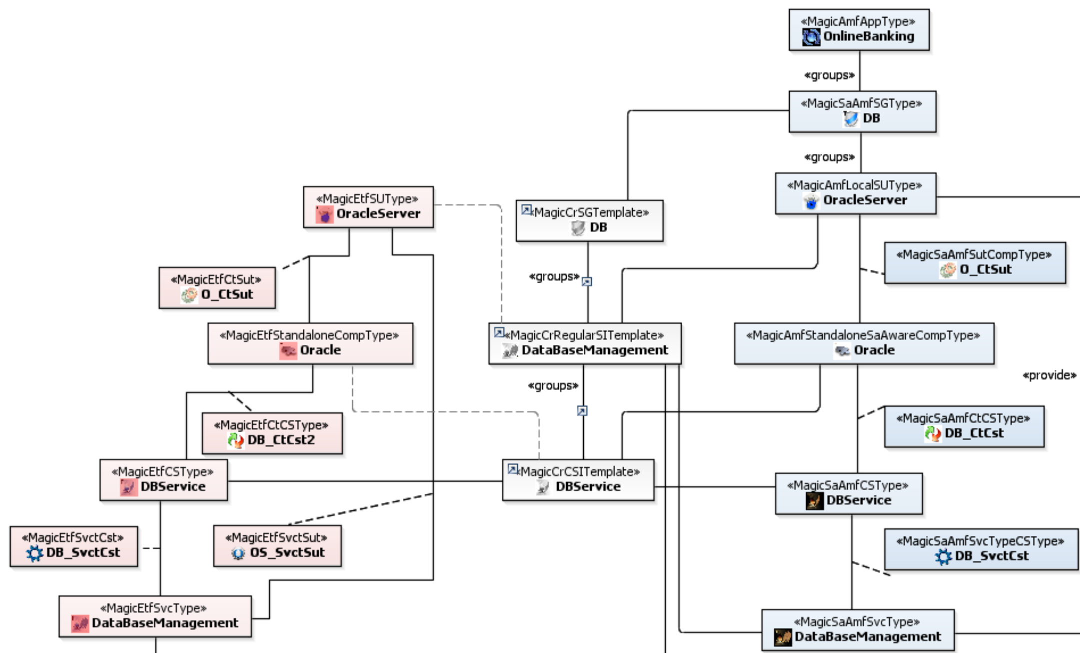

Figure 33 AMF Type creation phase for the DBMS part of online banking configuration

Another example of the AMF type creation phase for TransactionManagement SITemplate is presented in Figure 34, Figure 35, and Figure 36. Figure 34 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.

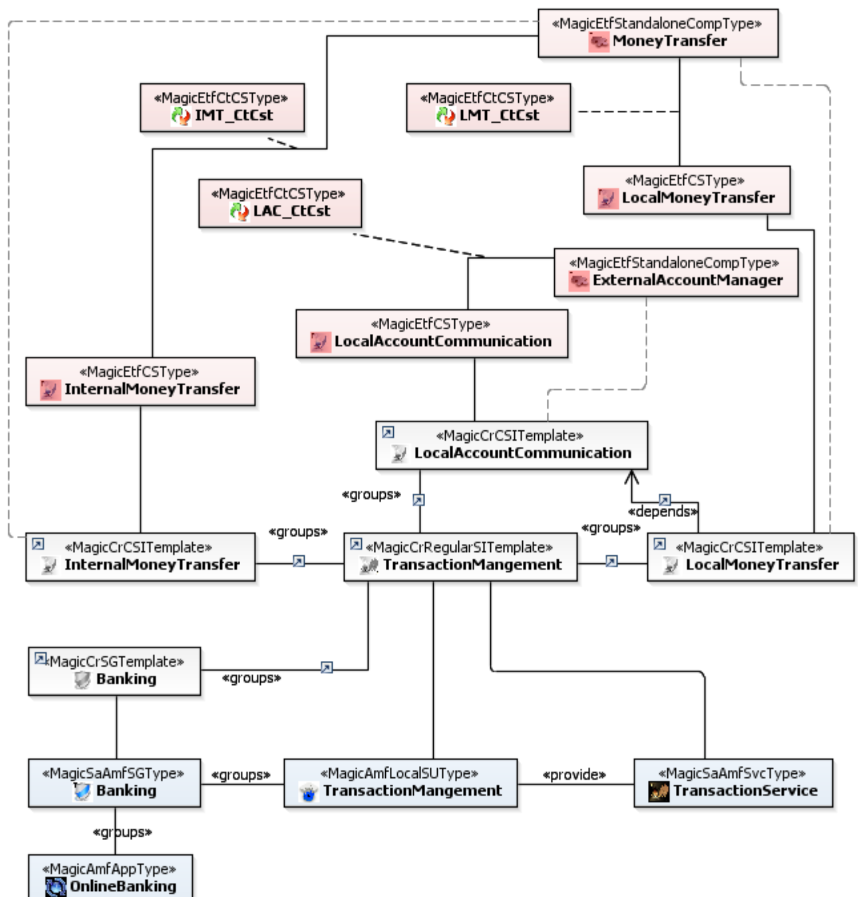

Figure 34 AMF SGType, AMF SUType, and AMF SvcType generation steps for TransactionManagement SITemplate and Banking SGTemplate

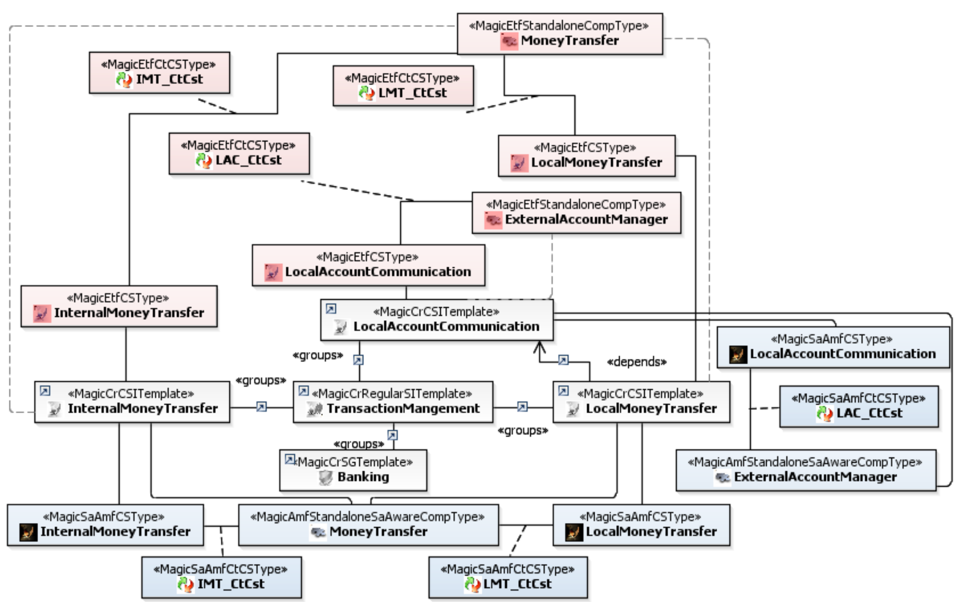

Figure 35 AMF Component Type and AMF CSType generation steps for the CSITemplates of TransactionManagement SITemplate

Figure 35 presents the result of the AMF Component Type and CSType generation phase (see Section 4.2.3) 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 35 uses the same names for both ETF types and their respective generated AMF types. Finally, Figure 36 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.

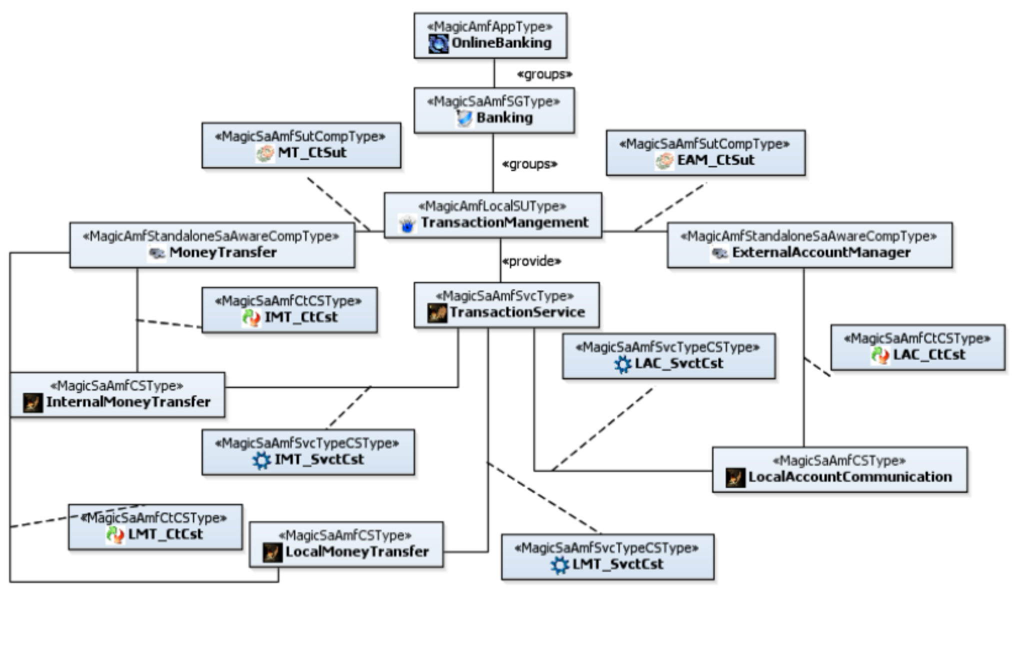

Figure 36 Created AMF Types for the transaction management part of online banking configuration

5.4.3 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 4.3.1, we specified the ATL rules for calculating the number of entities to be generated.

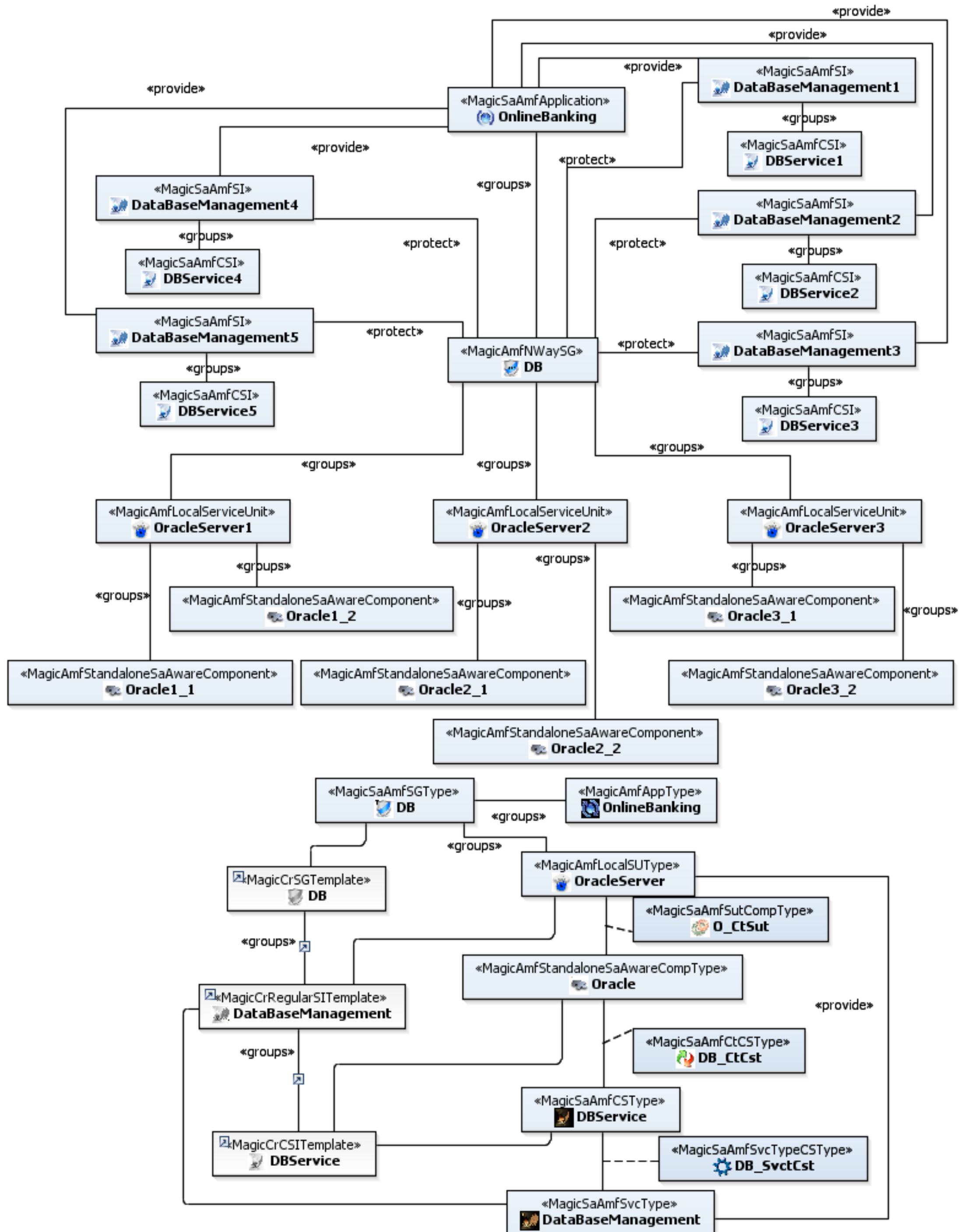

Figure 37 AMF entity creation phase for the DBMS part of online banking configuration

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 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 4.3.1, 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 37 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 4.3.3).

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.

6 Validation and Discussion

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 a 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.

Cabot et al. [Cabot 2010] 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~~ studies 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 to our model-based configuration generation approach which consists of a large number of transformation rules as well as complex input/output metamodels.

In this paper, ~~We~~ we ~~believe we have~~ followed a rigorous ~~and~~ stepwise process in designing the model-based approach. We ~~R~~reus~~in~~edg the knowledge gained during the specification of our modeling framework [REF[WH2]], which was validated by a domain expert.. with the objective of ~~certainly decreased~~ decreasing ~~the probability~~ errors in our approach. ~~Indeed, f~~For specifying

the transformations rules, we reused most of the OCL constraints specified in the AMF profile of our modeling framework [REF][WH3].

In addition, ~~Designing~~ 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. 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 2010] 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 Section 4, in each of the three main phases of our approach we store the selected/created elements that can be used to test each step individually. More specifically, in each step we checked if the transformation rules generated 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 our previous work [Salehi 2009 and Salehi 2011]. Although our validation approach is designed for the validation of 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 model-based configuration generation.

7 Conclusion

In this paper, we proposed a model-based approach for AMF configuration generation. The proposed approach is based on the model driven paradigm which has been shown to result in improved quality, serviceability, portability and flexibility.

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 [Salehi 2014 and Salehi 2015]. 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.

8 Future Work

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.

Acknowledgments

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

References

[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)

[Beizer 1995] B. Beizer, "Black-box testing: techniques for functional testing of software and systems", John Wiley & Sons, Inc., New York, NY, 1995

[Buskens 2006] R. Buskens and O. J. Gonzalez, "Model-Centric Development of Highly Available Software Systems", in Proceedings of International Conference on Dependable Systems and Networks, 2006, Philadelphia, PA

[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.

[Eclipse 2015a] Eclipse Foundation, Eclipse Model Transformation Project, URL: http://projects.eclipse.org/projects/modeling.mmt.atl, accessed October 2015.

[Eclipse 2015b] Eclipse Foundation, Eclipse Modeling Framework (EMF), URL: http://www.eclipse.org/modeling/emf/, accessed October 2015.

[Ehring 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.

[Hinrich 2004] T. Hinrich, N. Love, C. Petrie, L. Ramshaw, A. Sahai and, S. Singhal., "Using Object-Oriented Constraint Satisfaction for automated Configuration Generation". In: DSOM 2004. LNCS, vol. 3278, pp. 159–170

[Jouault 2008] F. Jouault, F. Allilaire, J. Bézivin and, I. Kurtev, "ATL: A Model Transformation Tool", Science of Computer Programming 72(1-2), 31–39, 2008.

[Kanso 2008] A. Kanso, M. Toeroe, F. Khendek, and A. Hamou-Lhadj, "Automatic Generation of AMF Compliant Configurations" in Nanya, T., Maruyama, F., Pataricza, A., Malek, M. (eds.) ISAS 2008. LNCS, vol. 5017, pp. 155–170. Springer, Heidelberg (2008).

[Kanso 2009] A Kanso, M Toeroe, A Hamou-Lhadj, and F. Khendek, "Generating AMF Configurations from Software Vendor Constraints and User Requirements" in Proceedings of the 4th International Conference on Availability, Reliability and Security (ARES 2009), pp. 454–461. IEEE, Los Alamitos (2009).

[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.

[Kovi 2007] A Kövi and D Varró, "An Eclipse-Based Framework for AIS Service Configurations" In: Proceedings of 4th International Service Availability Symposium, ISAS 2007 pp. 110–126.

[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.

[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.

[Mottu 2008] J.M. Mottu, 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 2011] Object Management Group. Meta Object Facility (MOF) 2.0 Query/View/Transformation, V1.1 formal/2011-01-01, URL: http://www.omg.org/spec/QVT/1.1/

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

[Piedad 2001] F. Piedad and M. Hawkins, "High Availability: Design, Techniques, and Processes", Prentice Hall, 2001, ISBN 9780130962881.

[SAF 2015] Service Availability Forum™, URL: http://www.saforum.org, accessed October 2015.

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

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

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

[Salehi 2009] P Salehi, F Khendek, M Toeroe, and A Hamou-Lhadj, "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 2011] P Salehi, F Khendek, A Hamou-Lhadj, and M Toeroe, "AMF Configurations: Checking for Service Protection Using Heuristics", in Proc. Of the 7th International Conference on Network and Service Management, Paris, France, 2011, pp. 1-8.

[Salehi 2014] P. Salehi, A. Hamou-Lhadj, M. Toeroe, and F. Khendek, "A Model Driven Approach for Availability Management Framework Configurations Generation", USPTO#: US8752003 B2, Patent filled 2011, granted 2014.

[Salehi 2015] P. Salehi, A. Hamou-Lhadj, M. Toeroe, and F. Khendek, "A Precise UML Domain Specific Modeling Language for Service Availability Management", Journal of Computer Standards & Interfaces –Elsevier, doi:10.1016/j.csi.2015.09.009.

[Sahai 2004] A Sahai, S Singhal, V Machiraju, and R Joshi, "Automated Generation of Resource Configurations through Policies," In: Fifth IEEE International Workshop on Policies for Distributed Systems and Networks 2004.

[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: 2nd Workshop on Middleware-Application Interaction, ACM (2008).

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

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

[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 Universit¨at M¨unchen, 2003 pp. 63-78.