Automatic Generation of AMF Compliant Configurations

Ali Kanso[1], Maria Toeroe[2], Ferhat Khendek[1], Abdelwahab Hamou-Lhadj[1]

[1] Electrical and Computer Engineering Department
Concordia University, Montréal, Canada
{a_kanso, khendek, abdelw}@ece.concordia.ca

[2] Ericsson Inc., Montréal, Canada
Maria.Toeroe@ericsson.com

Abstract. Service Availability Forum has defined a set of APIs to enable the building of off-the-shelf components for applications providing highly available services. A set of services has been defined and the Availability Management Framework is the service responsible of managing availability and therefore shifting this task from the applications to the middleware. Designing an AMF compliant configuration, for a given application, can be a tedious and error prone task because of the large number of attributes and parameters to be taken into account. In this paper, we propose an algorithm and the corresponding tool prototype for generating an AMF compliant configuration. We illustrate our approach with an example and discuss the main issues of the automatic generation.

Keywords: AIS, AMF, AMF Information Model, Entity Type File, Configurations Generation

1 Introduction

Service Availability Forum (SAF) [1] aims at providing high availability of network elements, systems and services through the usage of commercial off-the-shelf building blocks. High availability requires first of all no single point of failure, which is achieved by clustering, by use of different redundancy models, and by coordination of the resources within a cluster. SAF is developing and maintaining an Application Interface Specification (AIS) [2] for high availability middleware that is independent from any hardware platform and any specific vendor implementation. The SAF AIS defines the Availability Management Framework (AMF) to enable the management of the availability of services of applications that comply with the AMF information model and API [2]. Based on this information model, AMF coordinates the different resources in a cluster using the API.

The AMF information model describes the system configuration to be managed by AMF in terms of different software entities. Some of these entities characterize the service providers and their organization while others are related to the provided

services assigned dynamically by AMF to the service providers depending of their health and eligibility. The AMF information model also describes the types of these entities, as well as the cluster and its nodes where the entities are deployed.

The information model is provided to AMF through the Information Management Model service (IMM) [3]. An AMF compliant configuration to be loaded into IMM is described in XML (eXtensible Markup Language) [4] and accessed by AMF through the IMM service. A more formal and complete discussion of an AMF compliant configuration is provided in Section 2.

Developing a configuration in order to provide and protect services may be a tedious and error prone task to be undertaken manually by system developers. In this paper, we describe our approach for generating automatically an AMF compliant configuration from a set of type descriptors provided by the software vendor and from the configuration designer requirements, which include the service to be provided, its protection level indicating the redundancy model and the system to be deployed on. Obviously from a given set of type descriptors and a set of requirements, several AMF compliant configurations may be generated for the same system, which can be compared according to different criteria. In the approach presented in this paper, we are aiming at generating one AMF compliant configuration by integrating directly into the generation algorithm a certain number of design/configuration decisions. We discuss these decisions and their impact as they are encountered. We have implemented our algorithm in an ECLIPSE environment.

In the rest of this paper, we first provide the background knowledge on AMF configurations and related concepts. In Section 3, we elaborate on our approach for automatic generation of AMF compliant configurations from a given set of requirements provided by the configuration designer and a set of types describing the software coming from the vendor. We present the prototype tool and its application in Section 4. In Section 5, we discuss issues that have arisen during this research and future work.

2 AMF Compliant Configurations: Background and Related Work

2.1 Background

AMF is part of AIS, it defines a set of APIs for availability management through coordination of redundant resources [2]. In order to provide high availability, AMF requires a certain organization of the resources, i.e. a configuration, which is described by the information model. This information model consists of the different software entities to be managed by AMF in the running system in order to provide service availability, the types of these software entities that describe common features of the entities belonging to them, and the cluster nodes on which the software entities are deployed.

2.1.1 Software Entities

According to the AMF information model [2], the basic building block of an AMF configuration is a component. An AMF component is a set of software and/or hardware resources. Components are grouped into a service unit (SU) that combines their functionality to provide some services. In order to provide and protect services, SUs are grouped into service groups (SGs). An SG protects a set of services, which are represented as service instances (SIs). SIs are composed of component service instances (CSIs), when a particular SI is assigned to an SU, its composing CSIs are assigned to the components in this SU. The grouping of service groups forms an AMF application. Each SU is deployed on an AMF node, thus an SG is deployed on a node group. The set of all AMF nodes forms the AMF cluster.

AMF coordinates redundant entities (SUs and their components) according to a certain redundancy model that defines how many SUs (respectively components) are active, how many SUs (respectively components) are standby for protecting an SI (respectively CSI). For each SI AMF selects at runtime which SU shall perform in which role and makes the appropriate assignments via API callbacks to the components. Several redundancy models have been defined in the standard [2]. Each of them has its own characteristics. In the 2N model for instance, at most one SU can be active for all SIs, and at most one other SU can be standby for all SIs [2] the SG protects. A redundancy model may require specific component capabilities in order to protect the CSIs of the SIs. A component capability is defined as pair (x, y), where x represents the maximum number of active CSI assignments and y the maximum number of standby CSI assignments the component can have for a particular component service type. Therefore, depending on the redundancy model and the services to be provided, certain component types may be more suitable than others and some other component types may not be usable at all. For example, a component type that can have only active or only standby assignments at a time can be used in an SG that has a 2N redundancy model, but it is not valid for a SG that has an N-way redundancy model where each of the SUs may be active for some SIs and standby for others simultaneously.

Fig. 1 shows an example configuration of AMF entities. In this example, a cluster is composed of two nodes, A and B, with one SG protecting one SI in a 2N redundancy model. The SG has two SUs, SU1 and SU2, each composed of two components. The distribution of the active and standby assignments is decided by AMF at runtime.

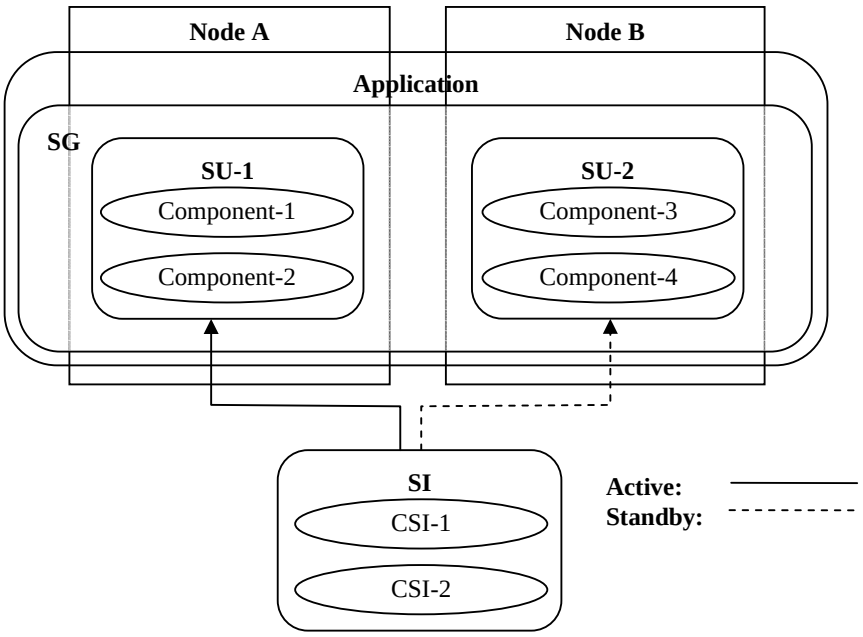

Fig. 1. Example of configuration of AMF entities.

2.1.2 Entity Types

There are two sorts of AMF entities: typed entities and non-typed entities. The typed entities are: the application (application type), the service group (SG type), the service unit (SU type), the component (component type), the service instance (service type), and the component service instance (CS type). The non-typed entities are: the cluster and the node. Although not shown in Fig. 1, each typed entity must refer to a type, since types are an integral part of the configuration.

Types are used in the AMF information model to define a set of common characteristics shared by all the entities referring to the same type. The entity types also determine the relation they have with other entity types. Thus, defining the relations their entities need to fulfill toward entities of other types. For example the SU type specifies the set of component types it contains, which defines components of what types must compose each of the SUs of the SU type. In the configuration generation algorithm in Section III, we pay particular attention to these types. The AMF entities and types and the relationships among them are described in the standard using a UML class diagram [5].

2.1.3 SMF View vs AMF View of Entity Types

The data required to configure these types, entities and their attributes, comes from two sources: The Entity Type File (ETF) [7] and the configuration designer. The SAF Software Management Framework (SMF) specification [6] standardizes the content

and the format of ETF [7] to allow software vendors to describe their products by means of types they implement. In this SMF context, types are used to characterize the software for all its possible deployments and settings. Hence, they differ from the types used in the context of AMF, which focuses on the runtime management aspects. Types of the SMF view could be perceived as meta-types to the types of the AMF information model.

ETF is an XML file provided by the software vendor to describe the software from SAF SMF perspective. For an AMF application implementation, it describes at least the component types and component service types implemented, including the dependencies and compatibility among these types. If there are further constraints on how and which the above types can collaborate to provide services, these constraints are specified in SU types, SG types, service types, and application types as necessary. As a result, the ETF may not be complete with respect to types for an application implementation, whereas in an AMF configuration all the AMF types must be present and complete.

Due to their purpose, there exist correspondences and discrepancies between the types described in an ETF and the types described in an AMF model. For example, AMF describes only the attribute names of CS types whereas in ETF in addition to this, the attribute type and range can be specified by the software vendor, to be able to configure the CSIs as AMF requires specific values for all the attributes**.** Similarly, dependency between component types may be specified in an ETF, but this is not reflected by the AMF information model since AMF does not need to know these dependencies among types. However, dependencies between components are captured in a different way. The instantiation level of components within an SU informs AMF to instantiate an independent component before those depending on it.

2.1.4 The IMM View of the AMF Information Model

AMF entities and types are represented as objects in the SA Forum Information Model [8]. The IMM service is the SAF service that manages these classes and corresponding objects. The SAF information model, including the AMF configuration can be described using a standardized schema, called the IMM XML schema [3]. An IMM XML file of a given configuration is loaded into IMM at system start up and made available to the SAF services including AMF through an API.

2.1.5 The Challenge of Generating an AMF Configuration

The goal of the configuration generation is to identify the set of service provider entities that matches the requested set of services and can protect them according to the requested redundancy model on a particular cluster. Generating such an AMF configuration is not straightforward. First of all, there are just too many inter-related entity and type objects that a configuration designer needs to work with. Consistency checks must be performed at various levels of the configuration generation process. For example, one must ensure that the ETF type dependencies and constraints are respected when creating components, assigning components to service units, etc. There are also key decisions that need to be made taking into consideration various constraints such as the maximum number of components of a certain type in an SU, the capability of a component when providing a certain component service type, etc.

In short, generating an AMF configuration could be a tedious and overwhelming task for a designer without tool assistance. Moreover, an automatic generation of the configuration will allow for the exploration of several potential configurations and compare and rank them according to predefined criteria.

2.2 Related Work

The standardization at SAF is ongoing, existing service specification are reviewed and updated as necessary, and more of the services are being defined. The B.03.01 version of the AMF specification on which the reported work is based differs significantly from earlier versions as it introduced the AMF types to be aligned with the first release of the Software Management Framework specification.

The work on implementing the APIs is ongoing in different places; OpenAIS [14]. OpenSAF [15] and OpenClovis [16] are open source projects aiming at developing a SAF compliant middleware for high availability. These provide limited if any support for automatic configuration design and none of them considers the AMF types yet.

The closest research work to the contents of this paper in the context of SAF has been reported in [10]. The authors in [10] apply the Model Driven Approach (MDA) to the design of AIS configurations. In this approach an initial AIS compliant configuration is devised using predefined design patterns, gathered from previous experiences. This initial configuration is referred to as the Platform Independent Model (PIM), which is then transformed and specialized automatically to a Platform Specific Model (PSM) to be used in a specific implementation of AIS. Meta-models are used for the transformation and for the validation of configurations. Our work is different from this approach, as we automatically generate this initial configuration or PIM.

More work on configuration generation has been done in the more general context of software configuration management, particularly using constraint satisfaction techniques and policies as reported in [11, 12]. Authors in [12], for instance, propose an approach for generating a configuration specification and the corresponding deployment workflow from a set of user requirements, operator and technical constraints, which are all modeled as policies. An example of constraints is, for instance, a given operating system can only run on certain processor architectures. Generating a configuration is formulated as a resource composition problem taking into account the constraints. Our approach is similar from this point of view; however, our focus is on the availability and AMF constraints instead of general utility computing environments. Challenging constraints, such as redundancy models to be provided, are not taken into account in [12].

3 Configuration Generation

In this section, we introduce and discuss the main steps of our configuration generation algorithm. In the current approach, we consider the generation of a configuration for only one AMF application consisting of SA-aware components

only, i.e. those that implement the AMF API. Thus, there is only one application type, and all other types are considered as subordinates to this type.

As shown in Fig. 2, the algorithm takes as input the ETF provided by the software vendor and the configuration requirements provided by the configuration designer including the application services to be supported. The next step focuses on determining the AMF entity types that can support these required application services: namely the SU types and the SG types. Once these types are determined, we proceed with creating their entities: components, SUs, SGs, as well as assigning SIs to SGs. Finally, the remaining entity and type attributes are completed. The generated configuration is specified in IMM XML.

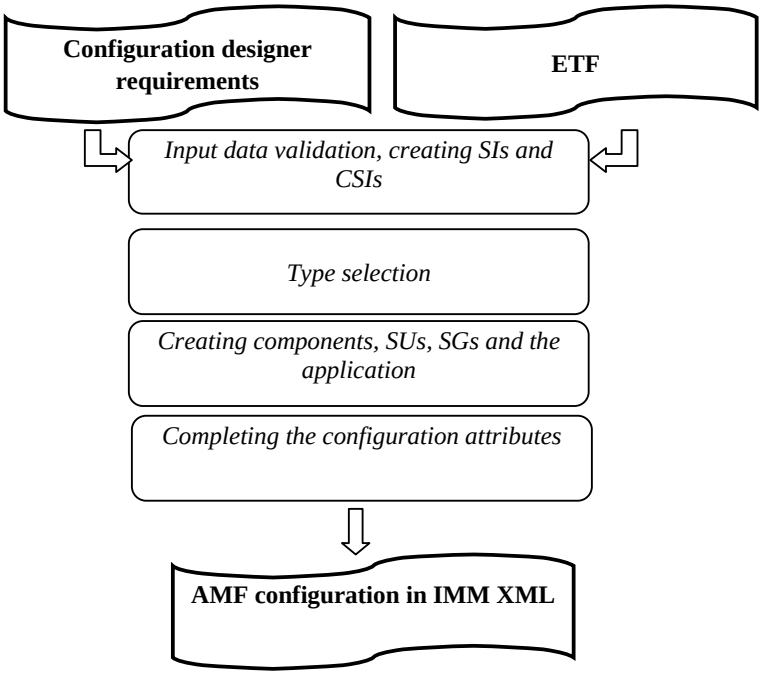

Fig. 2. Main steps for configuration generation.

3.1 Input Data and Validation

The algorithm takes three sets of input data:

- A set of ETF types that describe the software to be used,
- A set of services to be provided by the application, and
- A set of nodes on which the configuration has to be deployed.

As discussed in the previous section, the ETF types describe the software application from the vendor's perspective. This ETF must contain at least component types and CS types. Other entity types such as SU types, SG types, and the

application types may also be provided in order to capture limitations and constraints of the software. However, they are not necessarily provided in ETF.

The second set of input data is provided by the configuration designer and characterizes the services that the target configuration should support, i.e. what is the expected workload that needs to be handled. If any of the service types that needs to be provided by the target system is not specified by ETF, the designer must define them using existing CS types. The SIs and CSIs are specified for these service types. To facilitate this task the concept of templates has been introduced for specifying SIs. In an SI template, the configuration designer specifies for each necessary service type the total number of SIs to be created, within that the number of SIs to be assigned to the same SG, and the redundancy model of the SG that will protect these SIs.

Since SIs are collections of CSIs, each SI template is associated with a set of CSI templates based on the CS types defined for the service type. In each CSI template, the designer specifies the number of CSIs to be created for each required CS type within the SI. This must be in accordance with any constraints defined by service type, like the minimum/maximum number of CSI of a certain CS type that can be present in an SI.

Based on the SI and CSI templates the objects for the SIs and CSIs are generated and added to the configuration. The current solution does not cover the parameterization of CSIs. At this stage it is still a discussion topic.

The configuration designer may provide a cluster configuration, or an existing one can be used to obtain the information about the cluster (e.g. name, CLM cluster), and the nodes (e.g. number of nodes, fail over probation). Objects for these non-typed entities are created at this stage.

3.2 Type Selection

The objective of this step is to determine the types of the entities that will provide the required services as specified by the configuration designer. The primary goal is to match one of the existing types with the specified services. When no types exist for a particular compound entity, then an attempt is made to create the appropriate entity type. When no matching type can be found or created, because of constraints that cannot be met, then the software described by the types cannot provide the requested services as specified.

In the current approach, we assume that a SI template defines one SG type. All SIs of this template will be protected by SGs of this SG type. Within each of these SGs the SUs will belong to the same SU type. An SG protects SIs generated from one SI template only.

With these assumptions in mind, we start by determining the SU type that provides the service type of the set of SIs generated from the same SI template. If any SU types are provided in ETF, then we try to select one of them, and the selection is based on two criteria. The first one is that the SU type must provide the service type of these SIs. This is straightforward to check since an SU type specifies the service types it provides. The second criterion is that each SU belonging to an SG must be able to handle the load of SI assignments for both active and the standby states. An SU type may put restrictions with respect to the number of components the SU of this type can

contain and each of the components has its active/standby capability according to the component type specification. As a result an SU of an SU type may have capacity limitations that must be respected during the selection. For illustrating the constraint of load to assign to an SU, let us consider an SG with 5 SUs in an N+M redundancy model (with for instance 3 active and 2 standby). If, for instance, 60 SIs are to be protected by this SG, then each SU should have the capability of being active for 20 SIs or standby for 30 SIs.

The load each SU is expected to handle is calculated for the active/standby roles according to each redundancy model. It takes into account the number of SIs the appropriate SG shall protect, the number of assignments for each of the roles within the SG and the number of SUs among which this task is distributed within the SG at any given time. This gives the minimum number of SIs an SU needs to be able to handle in active/standby role. The calculated values are compared with the respective capacity of each SU type that can provide the necessary service type. The maximum active and standby capacities are calculated by first calculating the maximum number of SIs that the SU type can handle with respect to each CS type the SIs contain. Subsequently the minimum of these numbers is taken as the maximum capacity for the SI.

Several SU types that provide the required service type with the required capacity may be described in the ETF. On the other hand, it may happen that no such SU type exits in the ETF. In this case, the configuration generation fails. When no SU types are provided in ETF, i.e. fewer constraints are provided in the ETF, we build an SU type that provides the required service type with the required capacity. For the construction of this SU type, it may happen that more than one component types can be used. In the current approach, the preference is given to component types that have higher capability. The configuration generation will also fail, if based on the component capability no component type matches the redundancy model specified for the SIs.

Once an SU type is found or created, the algorithm proceeds to select from ETF an SG type based on the redundancy model specified in the SI template and which refers to the selected SU type. Note that many of the SG type's parameters were already used in the SU type selection. The algorithm terminates if none of the SG types matches. If no SG type is given in ETF then – as in case of the SU type – a new SG type is created using the selected SU type and the requested redundancy model.

The selection of SG and SU types is repeated for each SI template until all of them have been satisfied.

Provided that only types from ETF were selected in the previous steps, the algorithm applies the selection process to the application types specified in ETF. It selects the application type that references all the SG types that were selected previously. Otherwise it constructs a new application type as a union of the selected SG types.

3.3 Generating the Remaining AMF Entities

Once all the types have been determined and the corresponding configuration objects have been added to the configuration, we proceed with the creation of the

configuration objects for the AMF entities of these types. That is, the objects for SUs, components, SGs, and the application are created and configured.

In the SI template, the configuration designer specified – indirectly through the redundancy model – the number of SUs and SGs to be created, but not the number of components. To configure an SU, we need to determine this number. The number of components of each component type that must be created in an SU depends on the load, i.e. number of SIs assigned to the SU, the number of CSIs within those SIs, and the component capabilities. For an illustration, let us consider an SG that has 2 SUs protecting one SI composed of 5 CSIs of CST-A according to the 2N redundancy model. Let us assume the capability of the component that supports CST-A is 1 active and 10 standby. In this case, we need 5 components in our SU. While one component is enough to standby for all the CSIs, we need 5 components to be active for all the CSIs. .

The same calculation is repeated for every set of CSIs of the service type the SIs belong to. Thus, we populate the first SU of the first SG with all the necessary components. Subsequent SUs of the same SG are created by duplicating the first one and the SG is assigned a set of SIs that have been generated from the appropriate SI template at the beginning of the configuration generation algorithm as described in Section 3.1. Subsequent SGs for the same SI template are also created by duplicating the first one. The procedure is repeated for all SI templates until all SIs have been assigned to SGs.

3.4 Completing the Configuration Attributes

When the procedures described in Sections 3.1-3.3 have been completed, major part of the configuration attributes of the selected types and generated entities are determined based on the different selection and generation criteria described, however not all of them. To complete the configuration generation some additional attributes need to be assigned values. Here we briefly touch on a few that require further procedures.

If not specified explicitly in the configuration, an AMF implementation will decide on the assignment to nodes of the different SUs within the information model. The assignment procedure is not standardized, which means that different AMF implementations may do it differently. Our algorithm includes a procedure for distributing the generated SUs among the nodes of the AMF cluster and fill in the appropriate configuration attributes: The procedure assumes that the AMF nodes have equal capacities and therefore equally distribute the SUs among them.

In addition, we have developed a ranking procedure that enables equal assignment of SIs to SUs within an SG by completing the SU rank attribute for each SI. In other words, we ensure load balancing among SUs. Moreover, in the case of the N+M redundancy model, we rank SUs in a way that ensures that AMF would not replace an active SU that fails, with two standbys. The procedure for ranking SUs for SIs is dependent of the redundancy model attributes of the SG and the number of SIs.

4 The Configuration Generator Tool

The algorithm presented in the previous section has been implemented in a prototype tool developed in Java, using the Eclipse environment. It is anticipated to make this tool as an Eclipse plug-in to take full advantage of the capabilities of the Eclipse integrated environment.

4.1 Description of the Tool

Fig. 3 shows the overall flow of information. The user interacts with the tool through a Graphical User Interface (GUI).

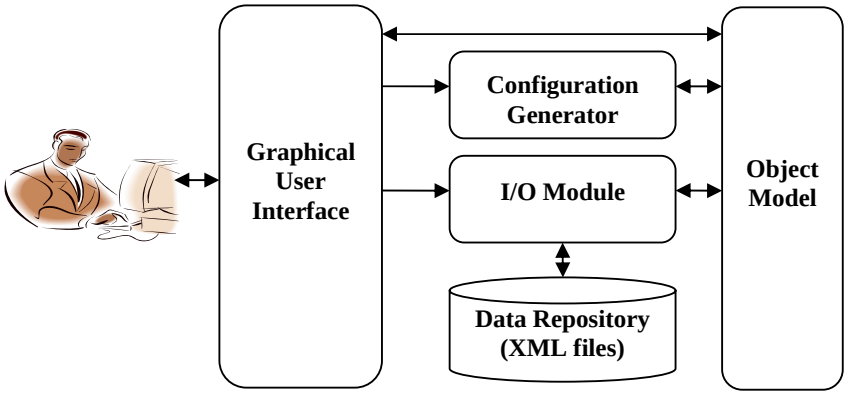

Fig. 3. The data flow diagram of the configuration generation tool.

The Object Model is based on the AMF information model described in the AMF specifications [2]. Additional classes and associations have been created to map the entity types defined in ETF schema [7] to the ones defined in AMF.

The Configuration Generator module encompasses the configuration generation algorithm presented in the previous section. It populates the AMF information model within the Object Model. The I/O module is used to save the configuration in IMM XML.

The I/O module also contains methods to read ETF and extract information from it. For this purpose, an ETF parser has been created. The data repository stores all data necessary for generating configurations including IMM XML and ETF.

4.2 The Tool User Interface

A snapshot of the prototype tool GUI is shown in Fig. 4; it consists of four views: The Input view (the left pane), the Attribute view (the middle pane), the AMF Instance view (the upper-right pane), and the Description view (lower-right pane). They are used to present the content of the Object Model from different perspectives.

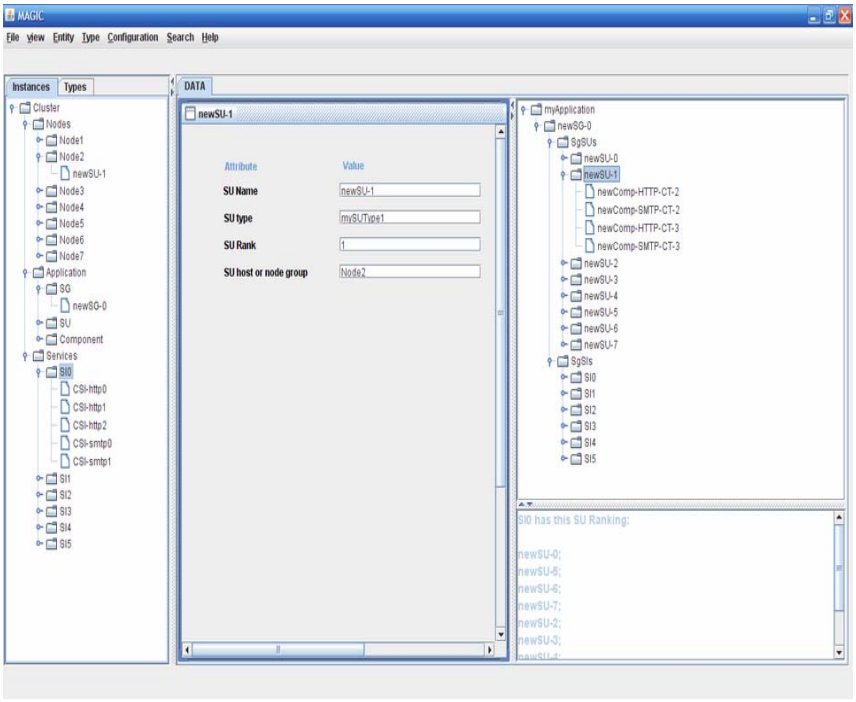

Fig. 4. Snapshot of the tool GUI.

The primary role of the Input view is receiving the input data for the configuration generation. Under the Types tab the AMF entity types read from ETF are presented to the configuration designer. It also allows for adding additional types that are not present in ETF yet the designer would like to consider for the configuration. Under the Instances tab the SI and CSI templates can be entered together with the non-typed entities. After running the configuration generation, this view will be completed with the generated entities and possibly new types.

The AMF Instance view is so called because it follows the structure of the AMF Instance View defined in the specification [2]. It contains the AMF entities of the resulting configuration after it has been generated. The Attribute view is used to display the attributes for the different objects selected either from the Input view or the AMF Instance view. Finally, the Description view displays in a textual form any additional information about the configuration which may not be present in any of the other views, such as the per SI SU ranking.

4.3 Application Example

To demonstrate the generation of an AMF configuration using the prototype tool, let us consider a simple example: A Web service application that provides e-mail

services using HTTP and SMTP protocols. Let us assume the ETF contains the following CS types: HTTP-CST, and SMTP-CST. It also contains the components types: HTTP-CT, SMTP-CT, and BAD-HTTP-CT.

Table 1 describes the CS types provided by these component types, as well as their capability models. Note that HTTP-CT and BAD-HTTP-CT provide the same HTTP-CST, but with different capabilities. Therefore, the tool has to choose from them. There is no dependency among the components. The service type for this example is created by the configuration designer and it consists of two CS types as shown in Table 2. The configuration designer is free to come up with any combination of component service types into service type to match the services of the target system as long as it satisfies the constraints imposed by the types described in the ETF. The tool verifies when the input is provided that such constraints are met.

As discussed in Section 3, CSIs (see Table 3) and SIs (see Table 4) are specified using templates. They are also provided by the configuration designer. They reflect the workload or traffic that needs to be handled by the target configuration and the desired protection level for these services.

Table 1. Component types and the CS types.

Component Type	**Provides CSTs**	**Capability**	**Dependency**
HTTP-CT	HTTP-CST	4 active and 3 stand by	None
BAD-HTTP-CT	HTTP-CST	1 active or 1 stand by	None
SMTP-CT	SMTP-CST	3 active or 2 stand by	None

Table 2. The service type.

Service type	**Member component service types**
Email-services	HTTP-CST, SMTP-CST

Table 3. The content of the CSI templates.

CSI template name	**Number of CSIs**	**CST**
HTTP-CSI-temp	3	HTTP-CST
SMTP-CSI-temp	2	SMTP-CST

Table 4. The content of the SI template.

SI template name	**Number of SIs**	**Service Type**	**Member CSI-templates**	**Redund. model**	**Number of SUs**
SI-temp	6	Email-services	HTTP-CSI-temp SMTP-CSI-temp	5 plus 3	8

The cluster's configuration is entered either by the designer or extracted from the current system configuration. This includes the number of nodes in the cluster, fail over probation, etc.

The configuration generated is displayed in Fig. 5. It contains one application that has one SG (NewSG-0) with 8 SUs (newSU-0 – newSU-7) as requested for the redundancy model in Table 4 (5 plus 3) and each SU has 4 components (newComp-HTTP-CT-0, newComp-SMTP-CT-0, newComp-HTTP-CT-1, newComp-SMTP-CT-1). Since 6 SIs need to be provided by the SG, this load should be split among 5 SUs for the active assignments and 3 SUs for the standby assignments. As it is decided at runtime, which SU is in which role, all of them have to be able to provide either role, which means 2 SIs per SU in each role. This means 6 HTTP-CST and 4 SMTP-CST assignments per SU have to be compared with the component capabilities. The tool gives preference to HTTP-CT based on its capability, which means 2 of such components are needed. The SMTP-CST is matched up only with SMTP-CT and also 2 of them are needed.

The size of the IMM XML file for this simple configuration is 85KBs. The size of this file for real life applications may be very challenging to be handled manually by developers.

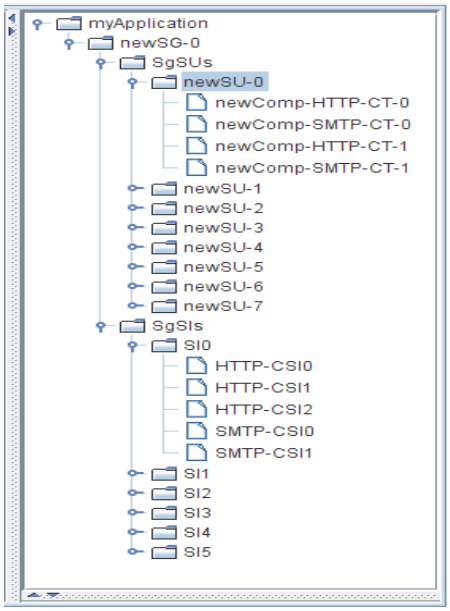

Fig. 5. A snapshot of the AMF Instance view for the example.

5 Conclusions

In the current approach, we consider the generation of only one AMF compliant configuration. This one configuration is created based on the strategy implemented in the generation algorithm during the selection or creation of different types, such as component types, SU types or SG types. However, using different strategies different configurations can be generated with a choice of alternative component types, SU types, or SG types. The criteria of selecting the types can change due to changing the preference for some of the attributes or in the future in case further description of the types are provided, such as the resources required by each entity of a specific type, of the mean time between failure for components, licensing cost, and many others as those taken into account in [13]. Having these attributes will allow us to generate multiple configurations according to different criteria and thus exploring a wider space of configurations and choosing the one that best suits the environment of deployment and the designer requirements. Moreover, we have considered the generation of a configuration for a single SA-aware application only.

It is important to mention again that there are situations where a configuration cannot be generated. This is the case, for instance, when the designer requirements do not match the system's hardware configuration. For example, this is the case when the designer requires hardware redundancy for an SG, but the number of SUs in the SG exceeds the number of nodes. While this may be easy to spot in some cases, in other cases it is not straightforward that the configuration designer requirements cannot be met with the given ETF and its constraints, that is, due to limitations of the software. For instance, one may find that there is no component type capable of supporting the required redundancy model; or the required load to provide the requested protection for the service instances exceeds the capacity of any service unit type provided in ETF due to limitations in the number of components or their capability; or none of the SG types in ETF uses the required redundancy model.

During the course of this project, we have encountered some challenges and came across some limitations of the AMF information model and ETF. The AMF information model will certainly benefit from some refinements and clarifications using inheritance, especially at the component and component type level. As for the limitations, one can mention that when more than one component can take a particular CS type in a given SU, there is no way to configure for AMF which component should be assigned. This becomes critical if there is a dependency between the component types and the CS types, which should be matched by the CSI assignments, but AMF has no information about it. The current form of ETF is not powerful enough to express all the dependencies/requirements a component type may have towards its environment (e.g. operating system, other applications). ETF allows for limited ways to define the valid combinations of component types. This is done through SU types and SG types. These combinations depend on the component types' interfaces toward each other, which is not captured in SAF.

This work is in progress and our approach is still evolving. This exercise helped us, the academic partners involved in the project, tremendously in understanding SAF specifications, the different aspects of AMF types (ETF versus AMF), the complex dependencies among the different types and their entities. We are working on improving our method in order to overcome its limitations, e.g. consider multiple

applications and from different categories as described in the AMF specification. We also plan to use more formal settings, such as the technique of constraints satisfaction as described in [11, 12], in order to explore efficiently all the potential solutions and compare them according to predefined criteria.

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

References

1. Service Availability Forum at: http://www.saforum.org
2. Service Availability Forum, Application Interface Specification. Availability Management Framework SAI-AIS-AMF-B.03.01.
3. Service Availability Forum, Application Interface Specification. Information Model Management Service SAI-AIS-IMM-A.02.01.
4. eXtensible Markup Language (XML) at http://xml.org
5. Unified Modeling Language (UML) http://www.uml.org
6. Service Availability**,** Forum. Application Interface Specification. Software Management Framework SAI-AIS-SMF-A.01.01.
7. SAI-AIS-SMF-ETF-A.01.01.xsd.
8. SAI-XMI-A.02.00.09.18.xml.zip.
9. SAI-AIS-IMM-XSD-A.01.01.xsd.
10. A. Kövi, D. Varró, "An Eclipse-Based Framework for AIS Service Configurations", Proceedings of the International Service Availability Symposium (ISAS), LNCS Vol. 4526, pp. 110-126, Durham, NH, May 2007.
11. T. Hinrich, N. Love, C. Petrie, L. Ramshaw, A. Sahai, S. Singhal, "Using Object-Oriented Constraint Satisfaction for automated Configuration Generation", 15th IFIP/IEEE International Workshop on Distributed Systems Operations and Management (DSOM), LNCS Vol. 3278, Springer, pp.159-170, Davis, CA, November 15-17, 2004.
12. A. Sahai, S. Singhal, R. Joshi, V. Machiraju, "Automated Generation of Resource Configurations through Policies", Fifth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'04), Yorktown Heights, New York, June 7-9, 2004.
13. G. Janakiraman, J. R. Santos, Y. Turner, "Automated Multi-Tier System Design for Service Availability", HP Laboratories Palo Alto, May 22nd , 2003.
14. http://www.openais.org/
15. http://www.opensaf.org/
16. http://www.openclovis.org/