An Eclipse-Based Framework for AIS Service Configurations

András Kövi[1,2] and Dániel Varró[1,2]

[1] Department of Measurement and Information Systems
Budapest University of Technology and Economics
H-1117, Magyar Tudsok krt. 2, Budapest, Hungary
kovi@mit.bme.hu
[2] OptXware Research & Development LLC.
H-1137, Katona J. u. 39., Budapest, Hungary
varro@mit.bme.hu

Abstract. In the paper, we propose an Eclipse-based model-driven framework to support an integrated development, analysis and deployment of Application Interface Specification (AIS) service configurations. Service configurations are first captured by platform-independent models (PIM), which directly correspond to the AIS standard itself, and abstract from vendor-specific details. Specificities of vendor-specific AIS middleware are incorporated into platform-specific models (PSM), which are derived from PIMs by automatic model transformations. Model analysis can be carried out either on the PIM-level to ensure standard compliance of a given service configuration, or on the PSM-level to detect availability bottlenecks by formal analysis early in the service configuration design. Finally, deployment descriptors of the selected AIS platform are generated from verified service configurations by automatic code generation techniques.

1 Introduction

As the range of business functionality is rapidly increasing to better meet customer needs, quality requirements are increasingly important in addition to rapid time-to-market development cycles. Availability, i.e. the continuity of a service, is one of the most important factors in the overall quality of business-intensive services.

However, in order to meet availability requirements, a service needs to be designed for availability by using well-founded development techniques. In order to avoid the re-development of best-practice solutions for achieving high availability, architectural-level solutions have been proposed based on best practices of constructing dependable systems.

The specifications of the Service Availability Forum. The *Service Availability*^{TM}*Forum (SAF)* [32] aims at providing standardized solutions for making services highly available. The *Application Interface Specification (AIS)* of the Forum defines the standard interfaces for accessing Highly Available (HA)

M. Malek et al.(Eds.): ISAS 2007, LNCS 4526, pp. 110–126, 2007.

middleware and infrastructure services that reside logically between applications and the operating system.

The entities defined in the AIS specifications (e.g. service units, message queues, applications, etc.) are described semi-formally by the *Information Model (IM)* in the form of UML classes [12]. The *Information Model Management Service (IMM)* [14] is the service in AIS that provides a set of APIs and administrative operations to create, access and manage the objects of the IM.

In the SA Forum ecosystem the *Software Management Framework (SwMF)* [15] provides all the functionality to migrate a system configuration to a desired new one. During the migration both the IM is updated and the required software entities are installed.

Problem statement. Although the classes in the IM define the common concepts that are used to build up SAF AIS compliant applications, standard descriptions do not alone guarantee that a certain service configuration will meet its quality of service (QoS) requirements. Moreover, the portability of service configurations between different commercial-off-the-shelf (COTS) AIS middleware implementations is problematic, i.e. deployment of a service configuration to a certain middleware makes it inappropriate for another AIS platform without changes, mainly due to the lack of standardization for the description of service configurations.

Objectives. In this paper we argue for the use of model-driven development techniques in the context of services to overcome the problems above. For this purpose, we present an Eclipse-based framework, which simultaneously supports the model-based development, analysis and deployment of SAF AIS compliant services. More specifically,

1. we propose modeling tools for describing AIS service configurations either by domain-specific modeling or using a UML Profile (see Sec. 3);
2. we present analysis tools to detect non-compliancy of a certain service configuration to the AIS standard, and highlight QoS bottlenecks in a configuration by using formal analysis tools (see Sec. 4);
3. finally, we demonstrate how automatic code generation and model transformation techniques can be used to derive vendor-specific deployment descriptors for service configurations (see Sec. 5).

2 An Overview of the Approach

In the following, we present an architectural overview of our framework to summarize its major components (see Fig. 1). Later, each of these components will be described in detail.

Modeling of AIS service configurations. We propose to adapt a model-driven development approach for AIS service configurations. For this purpose, *platform independent models (PIM)* of services are first constructed in accordance with the requirements specification. A well-formed PIM should conform

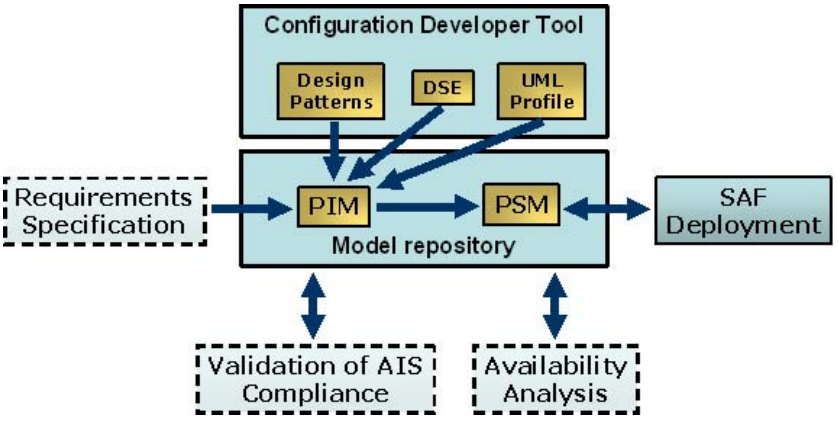

Fig. 1. Architecture overview

to the metamodel of the AIS standard itself. In a subsequent step, *platform-specific models (PSM)* corresponding to AIS middleware of a specific vendor can be derived automatically by model transformations. These models are captured either by a domain-specific editor for AIS models based on the *Eclipse Modeling Framework (EMF)*, or using a UML Profile for AIS services embedded into the off-the-shelf UML tool of IBM Rational Software Architect. Best practices of creating service configurations are grouped into design pattern libraries, which are made available to service architects.

Model analysis. The metamodel of the AIS standard can be complemented with constraints, defined in the Object Constraint Language (OCL), which capture additional well-formedness rules of the AIS standard in a formal way. When a PIM of a service configuration is available in the form of an EMF model, its conformance to the standard can be checked by validating the OCL constraints on the PIM (Sec. 3.2) using the OCL validation framework of EMF. Assuming that, HA parameters of a certain AIS platform are available for the designer, the PSM (Sec. 3.4) of a service configuration can be annotated with these service parameters. Based upon such an annotation, we can carry out formal analysis to detect availability bottlenecks early in the service development process by transforming the PSM of a service configuration into General Stochastic Petri Nets (Sec. 4.1), and analyzing different characteristics of those petri nets.

Service deployment. After analyzing a service configuration for a given AIS middleware, the actual deployment descriptors of the service configuration can be generated by automated code generation techniques, such as Java Emission Templates (JET) (Sec. 5).

The toolkit. This integrated model-driven development framework is based on standard, open interfaces as provided by Eclipse, and especially, the Eclipse Modeling Framework (EMF). The advantage of using Eclipse and EMF for the implementation is that there is a wide spectrum of tools that facilitate

development for the Eclipse platform, moreover, EMF has become the de-facto standard for model exchange in the industry nowadays. EMF is capable of generating the editor code and an example editor program for our metamodel that reduces the development time and the possibility of programming faults. In addition, EMF facilitates the validation of models, which will be described later in Sec. 4.

3 Modeling of Service Configurations

3.1 Requirements Specification

The service configuration development workflow starts with gathering requirements for service deployment. This information includes the type and number of components, the definition of services, service groups and the application itself. Since system resources for deployment are finite, priorities between services should be set up based upon the required availability for different services as part of the requirements specification.

There are well established schemes for defining the requirements of applications in specific application domains, however, it is still an open issue how these requirements can automatically be incorporated into the service configuration in the general case. Therefore, this phase of developing service configurations for AIS middleware is subject to future work.

3.2 Platform Independent Model (PIM)

In a model-driven approach, the development of service deployment configuration commences with the creation of a Platform Independent Model (PIM), which is the AIS configuration of the service. This PIM model of a service configuration is independent of the underlying platform implementation, thus it can be reused for different AIS platforms.

This PIM serves multiple purposes:

- it is used to integrate the service into the SAF ecosystem
- it is the input for the deployment procedure
- it can serve as input for generation of the source code of the service to speed up application implementation

Attributes and relations of conceptual AIS elements (service groups, service units, etc.) and other resources that are used by the service, for example, message queues and log streams, should be set up in this step.

AIS-PIM metamodel. The metamodel of the PIM (PIMM) of an AIS compliant application is built up from the entities defined in the specifications, thus, it ensures the compliance of the configuration model to the standard itself. As discussed in Sec. 1, the SA Forum Information Model (IM) contains a UML representation of service entities, e.g. it contains the Service Unit class that is

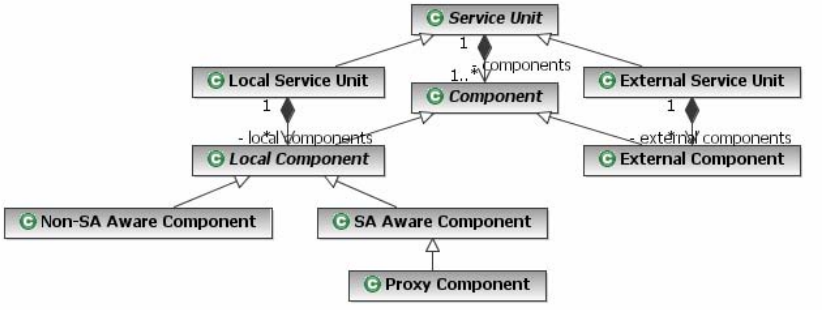

Fig. 2. Service unit and component class hierarchy

used to represent a service unit. Therefore, we have chosen the IM to serve as the basis of our PIM metamodel.

However, the Information Model, as defined by the SA Forum, is unable to identify certain semantic relations between service entities. For instance, issues like which service is the owner of a message queue or which log stream is opened by which application, are not represented in the IM model. Furthermore, for clarity purposes, we used the ontology listed in the AMF specification (Sec. 3.2 Logical Entities of [13]), which explicitly represents the type hierarchy of components and services, rather than the simple aggregated class concept used in the UML model of the IM. We believe that, the clarity of the AIS metamodel is highly improved by these changes.

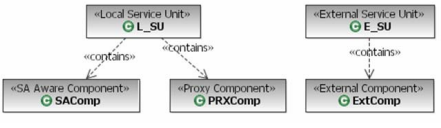

Fig. 3. Example model with stereotypes

The modified component and service unit hierarchy is depicted in Fig. 2. The general *Component* class is specialized into *Local Component* and *External Component* classes by classifying components according to the location of the component from the point of view of the *AMF cluster*. Then the *Local Component* is further specialized into *Non-SA Aware Component* and *SA Aware Component* classes. Finally, the specialized case of the *SA Aware Component* is the *Proxy Component*, which corresponds to the proxy components in AMF. On the other hand, the *Service Unit* class is specialized into two descendant classes: the *Local Service Unit* and the *External Service Unit* class. A *Local Service Unit* contains only *Local Components* while the *External Service Unit* comprises only *External Components*.

Because of the generality of the *Component*, *Service Unit* and *Local Component* classes, their usage in service configuration models is not allowed; and thus to avoid their instantiation, they are made abstract.

In Fig 3 a valid service configuration example is depicted. The class of the objects is indicated by the stereotypes, e.g. *Local Service Unit*, *Proxy Component*, etc.

Since this paper intends to give an overview of our framework and the techniques we use in it, only the most important changes to the metamodel are listed in the following:

- *Component class extended with reference attributes to all types of SAF resources* (e.g. message queue, lock, log stream) to support the indication of resource usage.
- *Runtime attributes*, that store the runtime state of the object, *are deleted from the classes* since such information is useless at design time. E.g. the attribute that stores the administrative state of a component or the one that contains the number of restarts of a service unit is removed.

3.3 Design Patterns

Building up the PIM manually from scratch can be a time consuming and error prone task. In an ideal case, a previously elaborated solution for a specific problem can be reused with changing some parameters. To help the developer in such cases design patterns are offered by our framework.

There are two types of design patterns for PIM development:

- **Fault tolerance related patterns** speed up modeling by providing parameterized procedures for automatic creation of ordinary objects and setting up their attributes,
- **AMF best practice patterns** are previously elaborated and stored solutions for more complex problems in certain application domains.

Fault tolerance (FT) related patterns help *create and configure all the necessary objects for a given fault tolerant architecture*. For example, if one creates a service group with 2N redundancy model then there will surely be at least two service units in that service group. Similarly, such "preconfigured" solutions can be provided to the user for all redundancy models defined in the AMF specification.

Another useful group of FT related design patterns are in connection with the topic of *software redundancy*. We talk about software redundancy when the simple multiplication of components in a service deployment does not provide sufficient fault tolerance, especially, against faults in the software. Such protection is essential for mission critical systems where erroneous behavior of a component can lead to catastrophic results. There are generally used patterns for these problems, e.g. *N-version programming (NVP)* [1] or *N self configuring programming (NSCP)* [23]. These redundancy schemes can be applied on SAF AMF managed software systems as well. As an example in the followings we show how the NVP scheme can be used.

In the NVP scheme several software variants compute simultaneously the same job/request, and when all of them are ready, a voter makes the decision on the final result.

Let us assume that we have three different software variants and want to use them in an NVP scheme (see Fig. 4). An SAF compliant service needs the following entities for this scheme: separate *Service Unit* for the *execution environment*,

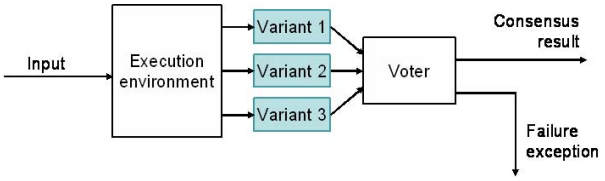

Fig. 4. NVP scheme for three software variants

for *each variant* and the *voter*, separate *Service Groups* for the *different functionality groups*, and finally, *message queues* are necessary for communication.

In Fig. 5 the AMF configuration for the described scheme is depicted. The *execution environment* sends the input requests through the *message queues* prefixed "*In_*" to the *respective variants* and the *variants* send their results to the *voter* through the "*Out_ queues*. (Note that, not all usage relations and AMF components are visible in the figure to prevent making it unnecessarily complicated.)

AMF configuration for all the previously mentioned software redundancy schemes can be defined in an akin way.

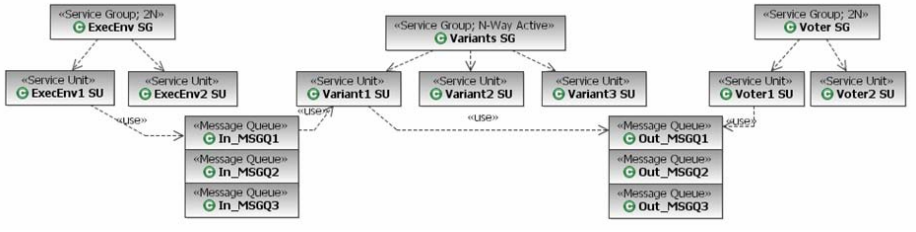

Fig. 5. AMF configuration for NVP scheme with three variants

AMF best practices. The other group of design patterns is best practices, which contain *previously stored system architectures and implementation related solutions for certain application domains*. System architectures can be for example, configurations for sensor networks or different systems that need a given level of reliability or availability. These best practices can have parameters as well that make them more flexible and more widely reusable (e.g. the number of sensors, data collectors, monitoring systems, etc).

3.4 Platform Specific Model (PSM)

Although the specifications of AIS standardize the interfaces an application may use to access the HA services of the middleware, there may be differences between platform/middleware implementations in the sense that how they are configured, and which services and functionalities are implemented. Thus, it is advantageous to create a *Platform Specific Metamodel (PSMM)* for each platform implementation.

Platform Specific Metamodel (PSMM). This metamodel *contains all the entities and their relations in the given platform implementation.* These entities may have extended or restricted features compared to the corresponding entities of the PIM metamodel. Furthermore, there may be additional entities for non standard services that a vendor-specific AIS implementation provides.

Using the notions of the PSMM the *Platform Specific Model (PSM)* of the service (*Service PSM*) can be created that describes a service configuration on the given platform implementation. However, the PSM of the service is preferably not created from scratch every time it is deployed to a different platform. If the PIM of the service exists it can be transformed into a PSM using a specific PIM to PSM mapping. The model transformation describes which element or elements in the PIM are mapped into which element(s) in the PSM model. (Technical details of such model transformations are discussed in Sec. 3.5)

3.5 Implementation Details

In the previous sections we described the configuration development workflow and its elements. A tool that facilitates this workflow has to provide a user interface that exposes all the required functionalities for creating, modifying, verifying, validating and transforming the introduced models. This user interface is called the modeling front-end.

The modeling front-end. UML is one possible language that we use to create service configuration models. UML provides a wide range of extension mechanisms (stereotypes, tagged values, etc.) to customize the basic language, and create domain specific dialects. A *UML profile* is the notion that encapsulates all the extensions of a specific dialect. We created a UML profile for AIS, i.e. a dialect, that contains stereotypes corresponding to the entity types of the AIS specifications, e.g. local service unit, service group, local component. An example for a stereotyped model created with IBM Rational Software Architect can be seen in Fig. 6.

Service models. Representation of the Eclipse Modeling Framework (EMF) [8] is the de facto industrial standard for storing and manipulating models in Eclipse. Metamodels in EMF are called Ecore models. The EMF model development workflow starts with *the creation of the metamodel* (i.e. an Ecore model). In our case it is the metamodel of the PIM and the PSMs. Then we *use the automatic code generation facility* that generates us the Java classes for the model, the model editor API, a sample model editor and class stubs for testing. EMF Ecore models are stored in XML files and this assures their easy reusability in other, non EMF based applications as well. EMF provides automated support for loading and serializing models from metamodel-specific XML formats corresponding to the XMI 2.0 standard. These EMF models can also serve as the basis of model validation as described in Sec. 3.

Domain Specific Model Editor. Domain specific model editors are generated editors, which are customized for specific application domains. In the Eclipse environment there are specific frameworks that provide means to easily develop

domain specific model editors. Most widely used and best elaborated is the *Graphical Editing Framework* (*GEF*) [9], which provides APIs for the creation of graphical editors. Since most editors provide the same functionalities, only the context and the outlook differs in many cases, the *Graphical Modeling Framework (GMF)* [10] has been started to support the development of rich domain-specific model editors. GMF provides means to define different aspects of the editor using specific models, and then automatically generate the source code for it.

Design patterns. Using the automatically generated specific model editor API, the modification of PIM and PSM models is possible from code. The design patterns library uses this AIS specific API to carry out a sequence of model manipulation operations. In our initial framework, design patterns are implemented as simple parameterized methods (without graphical user interface). Application of a design pattern on a model is done by calling the respective method with specific parameters.

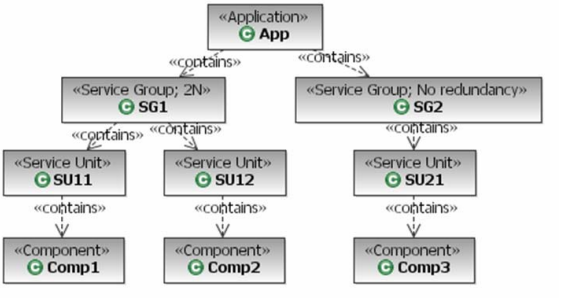

Fig. 6. Example stereotyped service configuration model

Model transformations. As mentioned above, model transformation is the mean that is used to generate the PSM from the PIM or the platform specific deployment descriptors from the PSM. The VIATRA2 transformation framework provides such model transformations by combining the formal paradigms of abstract state machines and graph transformation, which provide a rule and pattern based manipulation of models. As model transformations are out of the scope of this paper, here we do not describe them in detail. For more information on this topic see [34, 2].

4 Model Analysis

4.1 Static Analysis of Service Configuration Models

After creating a service configuration model, it is essential to verify its compliance with the AIS standard. This is carried out by formal verification of the model against constraints that are defined in its metamodel. These *well-formedness and semantic constraints* come from various requirements, e.g. *multiplicity restrictions* or *attribute values*, or a constraint may be composed of (i.e. it may refer to) *other constraints* as well. In Fig. 7 the types of model element

constraints and the direction of possible implications are depicted, while Fig. 8 shows an example for each type of constraint implication.

Object Constraint Language. For specifying constraints in object oriented models the Object Constraint Language (OCL) [26] of OMG [19] is a widely used standard formalism. It can be used to express additional constraints on metamodels that cannot be expressed, or are very difficult to express, with the metamodel itself.

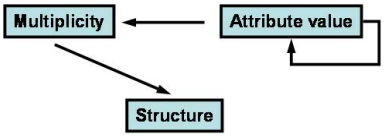

Fig. 7. Types of model element constraints. The arrows indicate the possible directions of implication.

Metamodel constraints. The OCL constraints are defined on the classes of the metamodel and their attributes. To ensure the correctness of the model we have to define constraints in the following cases:

1. *Value range restrictions for attributes.* (E.g. the size of the message queue (saMsgQueueSize attrib.) has to be greater than zero)
2. *Structural multiplicity restrictions.* (E.g. number of SUs in an SG)
3. *Attribute dependencies* where the value of one attribute depends on the value of some other attributes. E.g. if the component capability model of a component (saAmfCompCapability attribute) is *x_active_and_y_standby* then the maximum number of standby component service instances should be greater than zero (saAmfCompMaxStandbyCsi attribute).

Source	Destination	Example
Attr	Attr	The redundancy model of the service group (SG) (defined by the *asAmfSGRedundancyModel* attribute of the corresponding class) prescribes the required capabilities of a component. (E.g in a SG with N-Way redundancy model all components have to implement the *x_active_and_y_standby* component capability model.)
Attr	Multip	Redundancy model of the service group (SG) can define the lower multiplicity (i.e. the minimum number) of service units (SUs). (E.g. the 2N redundancy model supposes the existence of at least two service units.)
Multip	Struct	Each service unit of a service group should be deployed to different nodes in order to provide protection against node failures.

Fig. 8. Examples for different constraints

4. *Association dependencies.* E.g. service unit - service instance relations through the rankedSUs attribute.

In the following, we show two example OCLs on the PIMM. First, the simple example for *value range restriction constraint* is the *relative distinguished name* (RDN) constraint for *name attribute of the component* class, which describes that there cannot be two *components* with identical names in a *service unit*:

```
context ServiceUnit inv:
self.components -> forAll(c1, c2 |
        c1 <> c2 implies c1.name <> c2.name)
```

The respective part of the metamodel is depicted in Fig. 9. RDN constraints have to be stated for many other elements as well, like service groups, service units, etc.

A more complex sample OCL is the "*service types checking*" constraint for service instances:

```
context ServiceInstance inv:
if self.rankedSUs -> notEmpty then
  let requiredCSTs : Set<CSType> =
    self.csis.csType -> asSet() in
      self.rankedSUs -> forAll( su : ServiceUnit |
        su.components.csTypes ->
          asSet() -> includesAll(requiredCSTs)
      )
endif
```

The "*service types checking*" verifies that whether each *service unit* that the *service instance* is assigned to, by the ranked service units (*rankedSUs*) attribute, provides every *service type* the *service instance* requires. This constraint is a required condition for the successful assignment of the given *service instance*. The referenced part of the metamodel is depicted in Fig. 10.

Implementation. Checking of OCL constraints in an EMF based tool can be carried out by using the EMF OCL and Validation frameworks. At the time of writing the article there was no stable release of the mentioned frameworks, therefore,

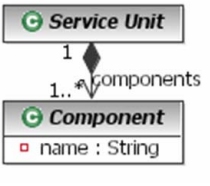

Fig. 9. Relation of service unit and component class

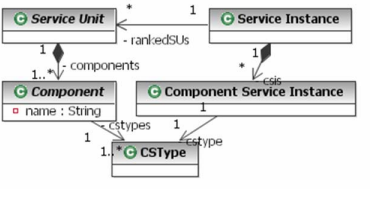

Fig. 10. Relation of service unit and service instance

most of the checks were actually (re)implemented in plain Java. However, the next EMF release, the Eclipse Modeling Framework Technologies (EMFT), promises to support the validation of OCL constraints over EMF models.

4.2 Non-functional Analysis

For users it is always a problem that the correctness of the configuration does not assure the appropriate functionality of the system.

Unfortunately, the compliance of a service configuration to the AIS standard does not alone guarantee that the quality of service requirements are met by the service configuration after deployment. For example, in high availability systems the expected availability and reliability of a service is a major parameter that has to satisfy certain required levels. Another problem is that during the installation of the new services some parts of the system may temporarily go down, however, the continuity of services has to be maintained in these periods as well, and this fact imposes constraints on the upgrade scenarios.

For these reasons, our framework supports *formal analysis of availability* as well. Often analysis techniques and tools can be integrated likewise.

Availability and reliability analysis. In HA systems the most important measures are the *availability* and *reliability* of the services. Standard dependability analysis techniques can be used to determine the value of these measures in a particular system to detect quality bottlenecks early in the design. In [24] the design, implementation and application of a tool is described that is able to *construct automatically a dependability model* (in the form of Generalized Stochastic Petri Nets) from a system architecture model. Then the *dependability model can be solved by an external solver* (e.g. the SPNP package [6]), computing in this way the system-level reliability or availability measures. The input for the tool is the stereotyped UML model of the system. In our case the PSM is adopted to the input of the tool chain by simple, rather syntactic model transformations, e.g. annotation by stereotypes, indication of usage dependencies, etc.

5 Automated Generation of Configurations

In the last phase of the configuration development we have to obtain the PSM of the deployed service configuration (*Deployment PSM* later), and then merge it with the *Service PSM*. The final step is the generation of the platform specific deployment descriptors using the *Merged PSM*.

In the following, first we discuss the solutions for reengineering a deployment configuration into a PSM, then we introduce different technologies and methods that are used for configuration generation.

5.1 Reengineering a Deployed Configuration

Before a new service could be integrated into an existing deployment the Deployment PSM has to be created. The following methods are available for reengineering the currently deployed service configuration:

1. Using model transformations the deployment descriptors are derived into a Deployment PSM
2. An IMM revealer agent traverses through the configuration tree and returns the Deployment PSM as the result of a request.

PSM creation by transformations. To eliminate the human faults, such as mistyping or misunderstanding the model, *model transformations are used for automatic generation of the Service PSM*. Moreover, the system deployment configuration files are not standardized, thus, *vendor specific transformations have to be written for each platform implementation*. Such transformations can be implemented, for instance, in the VIATRA2 model transformation framework [34] in the form of importer plugins and graph transformations. As model transformations is a complex topic and does not connect inherently to the subject of this paper, here we do not deal with it, but more information can be found in [24].

IMM revealer agent. In systems where the IMM service is available *a component can be written that traverses the Information Model and returns the Deployment PSM*. As mentioned in Sec. 3.5 Ecore models, and thus the PSMs as well, are stored in an XML format. As a consequence, *the agent simply has to return the PSM XML, which can instantly be used by the configuration developer tool* without any modifications.

5.2 Generating the Deployment Descriptors

The final step of the model-driven configuration development process is the *generation of platform specific deployment descriptors*. In Eclipse we can use *Java Emitter Templates (JET)* [11] for code generation from EMF models. JET is an easy to use and effective tool to generate the structured, platform specific configuration descriptor files automatically from the PSMs. JET templates take an object as input and produce formatted text using the different properties and attributes of the input object. These templates use a simple JSP [33] like syntax to describe the format of the output text.

In our case, the *Merged PSM is passed to the templates as parameter*, and the *resulting text is saved into a file*. Separate JET templates need to be created for each different output configuration file. An example JET template is listed in the following.

Listing 1.1. Example JET template

```
<%@ jet package="hello" class="GreetingTemplate"
                        skeleton="generator.skeleton" %>
 <%AISModel model = (AISModel)argument;
   foreach (SAFApp app in model.Applications) {%>
     <%=app.safApp%>,
   <%}%>
```

Alternatively, we could also use the code generation features of VIATRA2 for the same task.

6 Related Work

Model-driven development for Web services. The work presented in this paper was influenced by several proposals in different fields. First, proposals for the *model-driven development of service configurations have already been elaborated for Web services*, which have certain similarities with the SA Forum service configurations that we deal with, e.g. a complete framework based on the high-level modeling of Web services and their interactions with Web applications is described in [25]. In [18] a method is described for importing Web service descriptions into UML models, then integrating them, and finally generating the XML descriptors for the composite Web service. This process is similar to our approach to the integration of PSMs into a Merged PSM.

As we stated in Sec. 3.1 there are solutions for *integrating some specific requirements into service configurations*. As an example, a methodology is described for incorporating reliability attributes into Web service configurations in [17]. Furthermore, this process is carried out with model transformations using the VIATRA2 transformation framework. Other approaches that use model transformations for the integration of non-functional requirements can be found in [7, 31] and [22].

However, these solutions are different from our approach in that (i) only parts of the development process are supported and (ii) since Web service configurations have a standardized format the generation of platform specific descriptors is not an issue.

Non-functional model-driven analysis for services. We have used basic principles that are described in [5] and [24] for defining the model constraints, and carrying out the validation of the Service PSM models. Another solution for reliability prediction of a system based on UML models is described in [30]. They extend the Schedulability Performance and Time (SPT) UML profile [20] then perform analysis with transforming the UML model into a labeled transition system (LTS) using XML-based transformations. This solution sticks more to the standards based model-driven approach by using the standardized UML-XML mappings. However, the adaptability of the SPT Profile for AIS based services is unclear. Additional techniques for model-transformation based analysis of non-functional properties of service configurations are presented in [16, 3]. Finally, a method is introduced in [4] for generating optimal deployment configurations to a definite set of server nodes that guarantees the required availability and performance characteristics for all services.

Eclipse based configuration development tools. During our research we found only the OpenClovis IDE as an available Eclispe-based configuration developer tool for an AIS compliant platform,. The OpenClovis IDE [28] implements a subset of the functionalities that we proposed in this paper, e.g. creating and modifying AIS configuration models, generating source code and template based configuration development. The main difference between our proposal and the

OpenClovis IDE is that the latter implements only platform specific parts of the toolchain for the OpenClovis Application Service Platform middleware [27].

Our contribution. As a summary, the novelty of our approach compared to the previously enumerated works is that we define (i) *a complete model-driven methodology for service configuration development* (ii) *dedicated to the SA Forum ecosystem*, (iii) by developing a *toolchain using Eclipse-based technologies*. As a result, we defined a flexible toolkit that can easily be adopted to different needs of different platform implementations, meanwhile helping the developers with standard compliant platform independent model development for highly available service configurations. Even if some components of our framework are in an early prototype phase, we believe that the current paper provides relevant specification for future improvements.

7 Conclusions

In this paper, we presented an integrated model-driven configuration development method for AIS services and described a prototype toolchain that supports this process. Furthermore, we showed how such a tool can be implemented on the basis of Eclipse frameworks.

Although the specifications of AIS define the entities of the system and operations that an application may invoke, the format of the service configuration has not been standardized. Thus, the configuration of different platform implementations can be widely different. So as to support the modeling of the platform independent and platform specific views of the service configuration, we defined the Platform Independent Model and the Platform Specific Model. We use automatic model transformations for the PIM to PSM transformation as well, as transformations for model validation to speed up these processes, and to avoid human errors. Finally, the deployment descriptors for a given platform can automatically be generated from the PSM.

In the future we consider the following improvements:

- **Semantics-based model analysis.** Currently we define OCL constraints on the metamodel and then check them using the code generated by the EMFT-OCL framework. The problem with this approach is that, if a new constraint is introduced or an existing one is modified the validation code has to be regenerated or rewritten. Ontology-based model analysis provides a code-independent way for validating constraints. In ontology-based model analysis we define a formal ontology, which contains the metamodel as a T-Box (Terminology Box) and the model as an A-Box (Assertion Box). Then the ontology is passed to a reasoner like RACER [29] that decides whether the ontology is consistent or not. Simple OCL constraints can be implemented in the T-Box, while complex constraints are verified by using model queries. Such model analysis architecture provides more flexibility, however, it is restricted to a select of OCLs.

– **Integration of SwMF.** We think it is important to be able to generate standardized configuration descriptors as well (besides platform specific descriptors) that can be used by systems implementing the Software Management Framework (SwMF). Thus, we consider the development of a transformation for PIM that provides the essential *Entity types file*, which is used to describe the software entities that are delivered by a software bundle.

Acknowledgements

This work was partially supported by the HIDENETS project [21] of the European Union.

References

1. A. Avizienis. The methodology of n-version programming, 1995.
2. A. Balogh, A. Németh, A. Schmidt, I. Ráth, D. Vágó, D. Varró, and A. Pataricza. The VIATRA2 model transformation framework. In *ECMDA 2005 – Tools Track*, 2005.
3. A. Balogh and A. Pataricza. Quality-of-service analysis of dependable application models. 2006. Accepted for the 5th International Workshop on Critical Systems Development Using Modeling Languages (CSDUML 2006).
4. András Balogh, Dániel Varró, and András Pataricza. Model-based optimization of enterprise application and service deployment. In *ISAS*, pages 84–98, 2005.
5. Luciano Baresi, Reiko Heckel, Sebastian Thöne, and Dániel Varró. Style-based modeling and refinement of service-oriented architectures. *Software and Systems Modeling*, 5(2):187–207, June 2006.
6. Gianfranco Ciardo, Kishor S. Trivedi, and et al. Spnp: Stochastic petri net package - version 5.0.
7. Vittorio Cortellessa, Antinisca Di Marco, and Paola Inverardi. Software performance model-driven architecture. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 1218–1223, New York, NY, USA, 2006. ACM Press.
8. Eclipse modeling framework. `http://www.eclipse.org/modeling/`.
9. Graphical editing framework. `http://www.eclipse.org/gef/`.
10. Graphical modeling framework. `http://www.eclipse.org/gmf/`.
11. Java emitter templates. `http://www.eclipse.org/emft/projects/jet/`.
12. Service Availability^{TM}Forum. *Information Model Classes, SAI-XMI-A.01.01*, 2005.
13. Service Availability^{TM}Forum. *Availability Management Framework, SAI-AIS-B.01.02*, February 2006.
14. Service Availability^{TM}Forum. *Information Model Management Service, SAI-AIS-B.01.02*, February 2006.
15. Service Availability^{TM}Forum. *Software Management Framework, SAI-AIS-A.01.01.02 draft version*, 2007.
16. László Gönczy. Dependability analysis and synthesis of web services. In *Proc.* 13^{th} *PhD Mini-Symposium*, Budapest, Hungary, 2004.
17. László Gönczy, János Ávéd, and Dániel Varró. Model-based deployment of web services to standards-compliant middleware. In Immaculada J. Martinez Pedro Isaias, Miguel Baptista Nunes, editor, *Proc. of the Iadis International Conference on WWW/Internet 2006(ICWI2006)*. Iadis Press, 2006.

18. Roy Gronmo, David Skogan, Ida Solheim, and Jon Oldevik. Model-driven web services development. *eee*, 00:42–45, 2004.
19. Object Management Group. *Object Constraint Language specification.* `http://omg.org/technology/documents/formal/ocl.htm`.
20. Object Management Group. *UML Profile for Schedulability, Performance and Time Specification*, January 2005. `http://www.omg.org/technology/documents/formal/schedulability.htm`.
21. Highly DEpendable ip-based NETworks and Services. `http://hidenets.aau.dk`.
22. Henk Jonkers, Maria-Eugenia Iacob, Marc M. Lankhorst, and Patrick Strating. Integration and analysis of functional and non-functional aspects in model-driven e-service development. In *EDOC*, pages 229–238, 2005.
23. Jean-Claude Laprie, Christian Béounes, and Karama Kanoun. Definition and analysis of hardware- and software-fault-tolerant architectures. *Computer*, 23(7):39–51, 1990.
24. I. Majzik, P. Domokos, and M. Magyar. Tool-supported dependability evaluation of redundant architectures in computer based control systems. In E. Schnieder and G. Tarnai, editors, *FORMS/FORMAT 2007, the 6th Symposium on Formal Methods for Automation and Safety in Railway and Automotive Systems, 25-26 January 2007*, pages 342–352, GZVB, Braunschweig, Germany, 2007. ISBN 13:978-3-937655-09-3.
25. Ioana Manolescu, Marco Brambilla, Stefano Ceri, Sara Comai, and Piero Fraternali. Model-driven design and deployment of service-enabled web applications. *ACM Trans. Inter. Tech.*, 5(3):439–479, 2005.
26. Object Management Group. `http://omg.org`.
27. OpenClovis. Application service platform (asp), release 2.2. `http://www.openclovis.org/project/asp`.
28. OpenClovis. Openclovis ide. `http://www.openclovis.org/project/ide`.
29. Renamed abox and concept expression reasoner (RACER). `http://www.racer-systems.com/`.
30. Genaina Rodrigues, David Rosenblum, and Sebastian Uchitel. Reliability prediction in model driven development. In *ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems*, 2005.
31. Simone Rttger and Steffen Zschaler. Model-driven development for non-functional properties: Refinement through model transformation.
32. Service Availability^{TM}Forum. `http://saforum.org`.
33. Java server pages. `http://java.sun.com/products/jsp/`.
34. VIATRA2 Framework, an Eclipse GMT subproject. `http://www.eclipse.org/gmt/`.