# MODEL-DRIVEN APPROACHES FOR SERVICE-BASED APPLICATIONS DEVELOPMENT

Selo Sulistyo and Andreas Prinz
*Faculty of Engineering and Science, University of Agder, Grooseveien 36, N-4876 Grimstad, Norway*

Keywords: Model-driven, Development, Service-based Applications.

Abstract: Service-based systems are considered as an architectural approach for managing software complexity and their development. With this, a software application is built by defining a set of interactions of autonomous, compound, and loosely-coupled software units called services. Another way of managing software complexity is using model-driven approaches. With this, the development of software applications is started from model levels and thereby, code for implementing the software application is generated automatically. This paper presents AMG (abstract, model and generate), a combination of the two approaches.

## 1 INTRODUCTION

Information technology is spreading more and more into all areas of daily life, leading to an ever increasing amount of information and applications of a very high complexity. Already in 1968, (McIlroy 1968) has proposed the use of *software components* (units) instead of coding manually from scratch to build software systems (and applications). Based on this, several component models were introduced (i.e. modules, objects, components, and the last component models called services).

With software components, the development of complex software systems (and applications) is done in two phases. First, component developers develop components and second, application integrators compose them as new software applications. Here, the software composition is a way of the development of component-based software systems.

Another way of managing complexity of software systems is using models at different abstraction levels. With the idea, the development of software systems is done in an abstract manner on the problem spaces without taking much attention on the implementation details. In particular, OMG puts forward their idea of a model-driven architecture (MDA) (Kleppe, Warmer, and Bast 2003), which focuses on the software development using high-level models. With MDA, a complex software system is specified by models thereby code for implementing the solution can be generated automatically.

This paper presents AMG (abstract, model, generate), a model-driven method for developing service-based applications (SBA). With this, an SBA is built by defining sets of services interactions at model levels, then code for implementation is generated automatically. Note that we use the terms software system and application interchangeable.

The remainder of the paper is organized as follows: Section 2 gives a background for the paper. Then in Section 3, 4, 5, and 6 we present the AMG method, a case study, related work, and conclusion respectively.

## 2 MANAGING SOFTWARE COMPLEXITY

### 2.1 Software Component Approaches

With regard to the software component introduced by (McIlroy 1968), we consider that a service, which is a kind of an autonomous software unit, is the newest software component model that has evolved from the older component models (i.e. modular, object, and component models). They were developed for the purpose of managing software complexity and their development. In **modular systems**, for example, the development of a software system is split into small and independent modules that are developed and tested separately, that latter will be composed. Fort this, export/import is used.

Modular component models were considered inflexibility. To solve this, object-based systems



Sulistyo S. and Prinz A. (2010).
MODEL-DRIVEN APPROACHES FOR SERVICE-BASED APPLICATIONS DEVELOPMENT.
In *Proceedings of the 5th International Conference on Software and Data Technologies*, pages 288-291
DOI: 10.5220/0002926402880291

were introduced (Dahl, Myhrhaug, and Nygaard 1968). Software component models in object-oriented systems might either classes or objects. An object has a delegation function, inheritance, and aggregation that are considered as a way to compose objects to create new software systems.

The complexity of software systems is growing over time and since the modular and object-based systems were considered not to be able to manage the complexity, component-based systems were introduced. The basic idea is to completely isolate the implementation and to provide ready-to-use components. Among the most well known component models are OMG's CORBA, Sun's JavaBeans and Enterprise JavaBeans, Microsoft's DCOM and .NET.

Modules, objects, and components do not consider the architecture of the new created software systems. For this, service-based systems were introduced. A new application is built by defining set of interaction of loosely-coupled and autonomous software components called services. In general, service-based systems are well-known as a service-oriented computing (SOC) which promotes the service-oriented architecture (SOA) (Erl 2005) as an architectural technology.

The development of service-based systems is, in some ways, different from others older concepts (modular-based, object-based, and component-based concepts). The main difference from those older concepts is ownership and control. A service-based system does not have any control to the involved services. They can only use them by service invocations.

For the invocations, service description is used. Examples of XML-based service descriptions are services that are described using WSDL (Newcomer 2002), DPWS (OASIS 2009) and UPnP (Jeronimo and Weast 2003).

## 2.2 Model-driven Approaches

New ways of complexity handling take higher levels of abstraction and describe systems using models. In particular, OMG puts forward their idea of a model-driven architecture, which focuses on software development by means of high-level models. Using MDA implies creation of models of the following kinds: the *computational independent model* (CIM) the *platform independent model* (PIM) and the *platform-specific model* (PSM). The models (PIM, PSM) represent the structural and behavioral parts of a software system. From the models, *code* conforming to a specific programming language is generated automatically.

A model is an abstract presentation of a system which describes the structure and the behavior of the system. The structure specifies what the instances of the model are; it identifies the meaningful "components" of the model construct and relates them to each other. The behavioral model emphasizes the dynamic behavior of a system, which can essentially be expressed in three different types of behavioral models; interaction, activity diagrams, and state machine diagrams.

Expressing a software system by models requires a *modeling language*. The models itself might come in different abstraction levels. Here, the lower the abstraction level, the more information a model has. For these different abstraction levels, it might need to have different languages. For example, BPMN and BPEL (Allweyer 2009) is considered as a modeling language for the highest abstraction level (business models), which in MDA would be CIM.

However in software development, the main goal is always a running system. To achieve this, two alternatives exist. The first alternative is by developing an interpreter (or a virtual machine) that executes directly the models or the second alternative, by transforming the models into code conforming to existing programming languages using code generators.

# 3 THE AMG METHOD

AMG stands for abstract, model and generate. For the development of service-based system, defining service interactions by models is only possible if the service models are in place. For this, service abstraction is an important step. This mean also that service can be **concrete** or/and **abstract**.

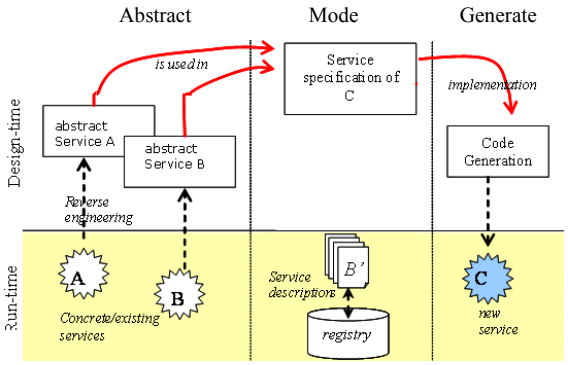


Figure 1: The AMG method.

### 3.1 Abstraction of Existing Services

See Figure 1. The abstraction process might be a kind of re-engineering, reverse engineering, or presentation processes. In the context of Web services, WSDL2Java from Axis (Goodwill 2004) is used to re-engineers service described with WSDL into Java classes. These classes are used as a proxy for the service invocations to real services in the service providers. With model driven approaches, these classes should be presented graphically, for example a UML class.

In this paper we do not use UML as modeling language instead of using Arctis (Kræmer 2007). With Arctis, a service is presented as a building block. For this, we have developed an abstractor to present services descriptions into Arctis building blocks. At the moment, the abstractor is able to read services with are described in UPnP and WSDL. The abstractor uses existing service frameworks, Cyberlink for Java for UPnP (Ouyang et al., 2007) and Axis (WSDL2Java) (Goodwill 2004) for the WSDL.

### 3.2 Modeling

We model an SBA as a building block diagram. Figure 2 shows a conceptual of a building block-based SBA.

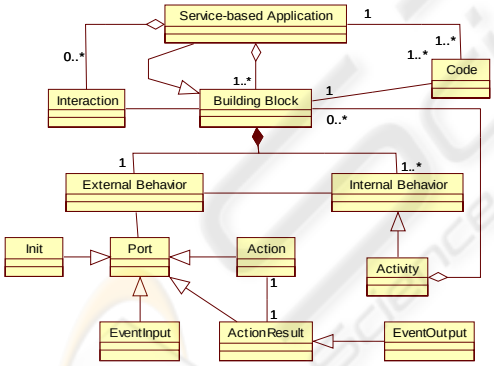


Figure 2: A conceptual model of SBA.

An SBA might include two or more building blocks that are connected using activity elements such as, relations, decisions, forks, and joins trough ports. A building block represents a service that can be considered as an encapsulation of activities (internal behavior). They execute functionalities. Ports are used to access this internal behavior.

### 3.3 Code Generation

The code generation process in AMG is depicted in Figure 3. In our prototype, one Java class will be generated for one SBA model (a system). The Java code for an SBA model is generated based on the models, templates, and proxy classes.

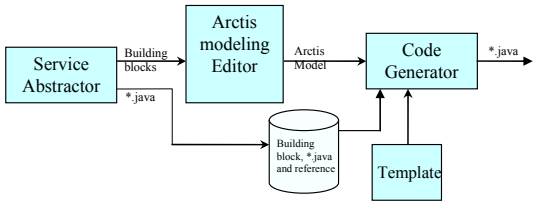


Figure 3: The code generation concept.

Different code generators can be used for different target languages. This can simply be done by applying a new template for each code generator. In our prototype we have developed template for J2SE code.

## 4 A CASE STUDY

To illustrate the AMG method we develop an SBA in a smart home environment where a residential gateway is controlling and managing home devices (and services) that running independently. Among of them are a *UPnP SunSPOT* service, *UPnP light* services and a *sendEmail Web* service. We want to provide an application and install it on the residential gateway that enables sending a message (email) and at the same time, switch on the light when the SunSPOT senses the home temperature is greater than 50 degree.

The first step of the development is abstraction of existing services into building blocks and implementation classes. Since we use UPnP and WSDL, the process is only a kind of presentation process, instead of abstraction. Each of building blocks has a reference to its own implementation class.

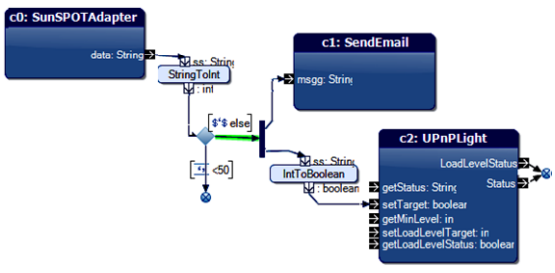


Figure 4: A model of the new SBA.

After all services are abstracted, then the models of the new SBA can be modeled. Figure 4 shows a model of the new software application as specified in the scenario. The model has include three building

blocks, a building block presenting the UPnP light service, a building block presenting the send email Web service and a building block presenting the UPnP temperature sensor (on SunSpot).

The behavior of the composite service is described using activity diagram. We consider that a building block is an entity that executes functionalities. Therefore, a model of collaboration of these entities using activity diagram is executable. The problem is how to formalize the activity diagrams as they are not formal enough. We will work on this in future work.

## 5 RELATED WORKS

Service composition is the only way for the development of SBA. SBA should able to include different service description technologies as a consequence of different ways of implementing service-based systems. Unfortunately, only a few method supports for non-Web Services technology, while there exists other service technologies that are potential to be included in the development SBA.

In Web service contexts, different techniques and composition languages exist. For examples, WS-BPEL (Ouyang et al. 2007) is considered as a composition language that can be used to compose Web services. However, BPEL is not a model-based language and for the execution, there is a need to develop/implement BPEL engines.

SOAML (OMG 2009) is also a language that can be considered as for services composition. However, SOAML is an UML profile and it is still a problem to automate the code generation from UML models.

The AMG abstract (or represent) existing services and present them graphically. The services are not limited for only Web services, but also other services (e.g. OSGi and UPnP services). AMG abstracts (or presents) existing services as building blocks, then based on the building block service integrators can specify building block interactions to define a new SBA. Since each building block refers to implementation class, code can be generated automatically.

## 6 CONCLUSIONS

We have proposed and demonstrated the AMG (abstract, model, and generate), a model-driven method for developing service-based applications. The prototyped AMG-abstractor abstracts existing services and presents them as building blocks. With these building blocks, a model of service-based software is created. At the end, code generators are used to generate code from the models. The AMG generates code fully automated, since each building block refers to its implementation class.

## ACKNOWLEDGEMENTS

This work has been supported by The Research Council of Norway in the ISIS project.

## REFERENCES

Allweyer, Thomas. 2009. BPMN - Business Process Modeling Notation. BoD, February. http://www.amazon.ca/exe%bidos/redirect?tag=citeulike09-20&path=ASIN/3837070042.

Dahl, Ole-Johan, BJørn Myhrhaug, and Kristen Nygaard. 1968. Some features of the SIMULA 67 language. In *Proceedings of the second conference on Applications of simulations*, 29–31. New York, New York, United States: Winter Simulation Conference. http://portal.acm.org/citation.cfm?id=805258.

Erl, Thomas. 2005. Service-Oriented Architecture (SOA): Concepts, Technology, and Design. Prentice Hall PTR, August.

Goodwill, James. 2004. Apache Axis Live: A Web Services Tutorial. Sourcebeat, December.

Jeronimo, Michael, and Jack Weast. 2003. UPnP Design by Example: A Software Developer's Guide to Universal Plug and Play. Intel Press, May.

Kleppe, Anneke G., Jos B. Warmer, and Wim Bast. 2003. MDA explained. Addison-Wesley, May.

Kraemer, Frank Alexander. 2007. Arctis and Ramses: Tool Suites for Rapid Service Engineering. In *Proceedings of NIK* 2007 (Norsk informatikkonferanse), Oslo, Norway. Oslo: Tapir Akademisk Forlag.

McIlroy, D. 1968. Mass-Produced Software Components. In *Proceedings of the 1st International Conference on Software Engineering*, 98, 88.

Newcomer, Eric. 2002. Understanding Web Services: XML, WSDL, SOAP, and UDDI. 7th ed. Addison-Wesley Professional, May. http://www.informit.com/store/product.aspx?isbn=9780201750812.

OASIS. 2009. Devices Profile for Web Services 1.1. OASIS. http://specs.xmlsoap.org/ws/2006/02/devprof/.

OMG. 2009. Service oriented architecture Modeling Language (SoaML): Specification for the UML Profile and Metamodel for Services (UPMS). Object Management Group.

Ouyang, Chun, Eric Verbeek, Wil M. P van der Aalst, Stephan Breutel, Marlon Dumas, and Arthur H. M ter Hofstede. 2007. Formal semantics and analysis of control flow in WS-BPEL. *Science of Computer Programming* 67, no. 2-3: 162–198. doi: http://dx.doi.org/10.1016/j.scico.2007.03.02.