Modelling Self-Adaptive Systems in Ubiquitous and Service-Oriented Environments

Paul Prescher
University Paderborn, Warburger Str. 100, D-33098 Paderborn

Abstract. Context aware self-adaptive component based software architectures are aiming towards systems, that can dynamically adapt themselves to changes in the context. These changes trigger adaptions via dynamic service replacement of the used services from the environment. The reconfigurations of the software are done with respect to the functionality and quality of service (QoS). In this paper a model-driven development approach is introduced to support the development of self-adaptive component based applications, named MUSIC. This approach is based on a planning framework and an UML profile. An example, that is related to a motivation scenario, shows the modeling and development approach.

Keywords: context awareness, ontology, self-adaption, component-based architectures, service-oriented architectures

1 Introduction

The mass of mobile devices and their effect on everyday life is increasing continuously. The environment of mobile devices consist of lots of services. Ubiquitous environments, in which the mobile devices are used, need reconfigurations of the applications during runtime, so that the best suitable service is used. This is because people are carrying the mobile devices around and using them for a lot of tasks in business and private life. The reconfigurations that are needed are changes e.g. when the battery is getting empty, the signal is getting weaker or new services appear or used ones disappear. To adapt the software, the regarding service is changed.

These self-adaptions of the software are leading to challenges in the development of mobile applications. In the past the context awareness of applications was not that important or simply not given. Now the development process has to deal with scenarios like unexpected changes and a wide range of infrastructure types for the applications. The problem of modeling adaptivity with domain-specific means is also discussed in [1].

MUSIC provides a framework to model context aware self-adaptive applications. These adaptions are done by dynamically by means of QoS properties. The self-adaption is done by changing components regarding the reconfigurations caused by the environment. This leads to a software with respect to the QoS, because it provides always the best available functionality even in case of context changes.
Further the developers do not have to care about the low level details of recon-
figurations.
This paper will introduce the basic requirements for a suitable solution. Fur-
ther the MUSIC project will be introduced which supports the planning of self-
adaption in mobile devices, with a view on service-based adaptions. The whole
development with MUSIC is a part of a MDA-based approach. The framework to
build the applications is component-based. The self-adaption and variability is
achieved by variation points during the design process. The modeling language
used in the tool is based on UML, this is done to profit from benefits of the
MDA approach. After the development the models can be used for automatic
code generation.

At first in this paper a motivation scenario is presented in Section 2. Then the
foundations will be introduced in Section 3. Here basic knowledge for the rest of
the paper will be provided. After that there will be an overview of the MUSIC
platform in Section 4, which includes the modeling techniques of MUSIC. After
that an example will be shown, which is based on the example from Section 2.

2 Motivation Scenario

The following scenario could be a possible scenario to motivate the need of
service-based adaptions. It describes the behavior of a mobile management ap-
plication that offers services like accessing business and customer information,
video calls and route planning. It always tries to support the user with the best
suitable service. This includes the kind of service as well as the quality of the
service (bandwidth and so on).

**Scenario:** The manager of a big software company is in a meeting at a different
company. This company is far away from the next big city, the mobile internet
is really slow. Because of this the mobile management application switches to
the local Wireless-LAN.
The mobile management application receives a video call from another customer,
so the manager takes it. During the call he has to leave the building and walks
towards the train station. The Wireless-LAN signal is getting weaker and because
of this the bandwidth is decreasing. Thus the management application switches
to the UMTS network. The change of the network triggers some reconfiguration
so that the responsible component is exchanged. The UMTS network cannot
support the full bandwidth. Because of this the management application has to
reduce the resolution of the video call due to less bandwidth. Further it uses an
offered service for navigation, which is offered by the UMTS network.
When he reaches the train station he changes the navigation service to an offered
service by the local Wireless-LAN to navigate him to his train.
To support the work the mobile management application is switching into an
"non-intranet" mode during the train ride. This means that the manager has
no access to the company data, only the data stored on the mobile device can
be used. When he reaches the company the mobile management application switches to the local company Wireless-LAN. The Wireless-LAN offers services to support him with information about his company, so the "non-intranet" mode is changed to "intranet" mode. This offers further information about the working employees, meetings and so on.

3 Foundations

In this section foundations and basic definitions will be introduced. This knowledge is important and will be used in the further discussion of this paper. It includes three main topics: Model Driven Architecture, Component-based architecture and Web Ontology Language for Web Services.

3.1 Model Driven Engineering (MDE)

Model-driven engineering (MDE) is a software development method with focus on creating domain models. These models are an abstracted representation of the knowledge and activities of an application domain. With the MDE approach the productivity and the compatibility to other systems can be increased because of the possibility to reuse standardized models. MDE is standardized and uses formal modeling languages such as UML.

Different models are used for different views on a project, such as class diagrams as representation of the structure and sequence diagrams for the communication. Models can be modeled using graphical or textual modeling languages. The models enable to understand complex applications by reducing the complexity and by providing different views on the problem domain. The reducing of the complexity is done via an abstraction from the details that are not needed for the modeling.[2]

3.2 Model Driven Architecture (MDA)

Model Driven Architecture (MDA) is a model driven engineering approach of the Object Management Group (OMG) that focuses on the possibility to generate implementations for different platforms out of models. This increases the platform independency.

The goal of MDA is to decrease the development time and the complexity in form of abstraction for a better handling. The models that are developed during an MDA process are easier to understand for humans. Better understandability offers the opportunity of easier communication on implementation details.

The possibility of generating code from the models that are produced during the development process leads to the advantage, that models are better understandable by humans. Errors can be avoided that depend on human failures in case of programming, as well the development time can be decreased. [3] MDA is a specialization of Model-Driven Software Development (MDSD) in the sense that:
A domain specific language (DSL) is defined on the basis of the meta object facility (MOF)\(^1\). This allows every notation of a meta-model as long as it is defined over the OMG-Metamodel.

The use of UML-Profiles is recommended as a concrete syntax for the DSL. The static semantic is specified with OCL-constraints.\(^2\)

Different views on formal models are defined. This includes for example a platform specific (PSM) and a platform independent model (PIM).

For model to model transformations in MDA query/views/transformations (QVT)\(^3\) is recommended by the OMG. This is a standard of the OMG.

MDA has the possibility to create an action semantic which allows to specify algorithms in an abstract way. This can be used to program like in normal programming languages in corporation of the static model.\(^7\)

3.3 Component-Based Architecture

Component-based architecture is a paradigm to develop software. It uses the concepts of software components. This increases the reusability of software artifacts.

The basic idea of component-based architectures is the partition of the application into components. These components should be reusable, this has the big advantage that the regarding code can be reused and does not have to be implemented again.

A component is defined as a software package that provides a set of functions. The components have interfaces, which are used for the communication with other components.

When some components are developed, the application can be built via choosing different components which fulfill the requirements of the application. Furthermore components only have to be developed for the missing functionality.

Advantages of the component-based architecture are \(^8\):

\begin{itemize}
  \item Less development time
  \item Higher Quality (more users and usage scenarios lead to automatic test cases)
\end{itemize}

In Figure 1 is an example of a component based structure. This consists of three components, UI component which is representing the user interface, VideoComponents which provides data from video calls for the UI and BusinessComponent which provides data from other business components for the UI. Now the Video-Controller can be exchanged, the if the regarding component fulfills the interface.

\(^1\) The Meta Object Facility (MOF) describes a special metadata-architecture, designed by the OMG \(^4\)

\(^2\) Object Constraint Language (OCL) is textual specification language for invariants \(^5\)

\(^3\) QVT is a language to support model to model transformations, developed by the OMG \(^6\)
3.4 Web Ontology Language for Web Services (OWL-S)

In general an ontology is a formal representation of knowledge as a set of concepts within a domain plus the relationship between those concepts.

The Web Ontology Language for Web Services (OWL-S) is a specification to describe Web Services. It specifies ontologies to describe a Web Service on a technical level. This contains the precondition to execute the service, the input, output and the side effects of the service.

The goals of OWL-S are:

- **automatic Web Service Discovery** (automated process for location of Web Services that provide service capabilities)
- **automatic Web Service Invocation** (automatic invocation of a Web Service by a computer program)
- **automatic Web Service Composition and Interoperation** (automatic selection, composition and interoperation of Web Services to perform some task done by a given high level description of an objective)

OWL-S provides three different types of knowledge. Each type of knowledge answers a different question. The service profiles answer the question "What does the service provide?", the service grounding "How does it work?" and the service model "How is it used?". [9]

A related approach is discussed in [10]. In [11], an approach how different services can be combined to reach a certain goal is discussed.
4 Middleware Support for Self-Adaption in Ubiquitous and Service-Oriented Environments (MUSIC)

In this section the "Middleware Support for Self-Adaption in Ubiquitous and Service-Oriented Environments" (MUSIC) project will be introduced. First the requirements for a solution will be presented. Afterwards the MUSIC platform will be introduced. Subsequently some possible enhancements are shown. At the end a modeling example is presented.

4.1 Requirements to a solution

Self-adapting context aware solutions need to fulfill special requirements in order to reach their functionality. They have to use a variety of services to adapt to the current situation of the environment. Often these adaptations depend on external services. These external services have to be found, the software needs the functionality to change regarding to this service and the correct service has to be identified. Further the approach has to follow the MDE. This leads to the following requirements [12] [13]:

Application variability: In order to reach dynamic reconfigurations a model is needed to define application variability by the developer. This means that the developer needs to specify when and where components can be changed and based on which reasons these changes occur. Since these changes are brought about to reach a better QoS, the model to specify these adaptions has to be enhanced by metadata for the QoS of the involved components. Further the software needs a mechanism to decide when to change. This has to be done in order to achieve the best provided service, based on agreements for the given quality of the service (QoS) and the violations of these agreements.

Dynamic discovery of services: As mentioned before services to which the software can adapt are discovered dynamically based on the current environment. This dynamic access during the runtime needs a special mechanism to discover the best service. This information has to be integrated into the application variability model. In order to support the dynamic service discovery correctly a semantic description of the QoS properties is needed.

Heterogeneity: Because the services in ubiquitous computing environments are developed by several independent developers, the QoS properties and the context information, used to describe the execution context have to be specified in a heterogeneous way. This will be especially needed if third party context sensors are integrated. The mentioned properties have to be semantically enriched to support the interoperability and integration of the different components. To reach heterogeneity, a comprehensive modeling notation is needed. A common vocabulary enables a shared semantic view on the involved components and services and the execution context.
Integration of service and context properties into the planning: The usefulness of alternatives of the application is part of the adaption planning. Under given context situations it has to be evaluated which service is the most useful one. In order to reach this, the QoS properties of services and components have to be associated to the properties of the current execution context.

4.2 MUSIC

In this section the core concepts of the MUSIC platform will be introduced. The modeling notations provided by MUSIC and a complete example will be given.

The goal of the MUSIC approach is to provide the best service with the best quality to the users. This happens based on the context of the user and the users preferences. To reach this the software has to be aware of context changes and adapt itself regarding to them. Thus alternatives have to be derived that can be used to provide the best quality for the adoptions.

As shown in Figure 2 MUSIC consists of a Conceptual Component Model which is used to describe an application represented by component types and their interfaces and dependencies. The arrow from the Conceptual Component Model to the Component Framework represents that the components are used for realizations of the application variants inside the Component Framework. This specifies the possible reconfigurations. Composite Realization Plans are describing a certain composition of components. The MUSIC Ontology enhances OWL-S in some parts to support the semantic service discovery, description of service levels and service level agreements. Further the output of the ontology is annotated to the model elements. [13] As shown on the right side of Figure 2 the middleware automatically creates variants of the application. This happens by information of the context. A utility value and property constraint are computed, then the best suitable adaption is selected for the reconfiguration. [14]

![Fig. 2. Overview of the MUSIC project](image-url)
**Conceptual Component Model:** This approach is based on components and the variability is achieved by variability points. These are realized in the ComponentType and Plan.

A component is a unit of composition. Interfaces and dependencies are specified by interfaces and the execution platform.

The meta-model of the conceptual component model can be seen in Figure 3. In this meta-model a component has one component type. One component can consist of components again. Each component type has different realizations and provides and requires other services. This is done via ports. A port type has a set of required and provided interfaces. These are characterizing the port type. Sets of port types represent different characteristics of the different component type and its connections.

By specifying the QoS properties for each port type, component contracts are specified. Constraints are added to associate concrete values with the properties of a component.

Components can have different properties. There exist SoftwareComponents, Resources and UserEnvEntities. SoftwareComponents can use other software components or resources. Resources represent available resources that can be used by other components during runtime. The properties can express for example, that its capacity is limited. The UserEnvEntity models the entities of interest in the environment of the user (light, noise [14]. This environment can affect the resources.

Fig. 3. Conceptual component model
In the development process components and component types are considered as classes. [14] Variations are described using Composite Realization Plans. These describe the structure of a component type by specifying the components and connections. For the variation one of the different plans of one component type is selected. The plans are solving the problem of the integration of service and context properties into the planning. [12] The components are used in the Component Framework to specify possible reconfigurations. All realizations of the the Conceptual Component Model represent the Component Framework [12].

Component Framework: The Component Framework is based on variants of components providing functionality, these define the functionalities that can be changed dynamically. All realizations of the component types represent the component framework.

Changes of the components which are responsible for the functionality are chosen by a utility which was computed before. This utility of alternatives in respect to context changes is evaluated by the built component framework based on some utility function specified before. These points of alternatives are called variation points. They provide different components which represent the same functionality but differ in the achievement of the functionality(e.g. via WLAN or UMTS). [14] This part is responsible to solve the application variability mentioned in the requirements.

In Figure 4 the component framework can be seen. Here an application type is

![Component framework diagram]

Fig. 4. Component framework

a component type. This is seen in Figure 3 realized by components. The compo-
Components can be specialized. As specialization they can be either atomic components or composite components. How the component type is realized is described by plans. There are two types of plans, composition plans and blueprint plans. Atomic components are described by blueprint plans. Composite components are described by a composition plan. It specifies the involved component types and connections between the component types. One plan is one realization of a component type. Now different plans can be specified to describe the variation of a component type.

Because the application types are represented as components, there is no need to specify all possible application variants. The plans can be used to create variations of the application using different components. Only the overall structure of the plans has to be provided. One composition plan can lead to different application variants. This plan specifies the cooperating component types. These can again have different realizations. These realizations have their own plans again.

**UML-Profiles:** UML-Profiles are used to support the modeling of application types, component types and to realize plans. The targets of dynamic service discovery are annotated by a set of descriptions specifying the expected functionality, interfaces and QoS properties. To model the application architecture UML 2.0 is taken. All entities of the application are modeled and linked by an enhancement of the UML 2.0 superstructure.

**MUSIC Ontology and QoS Enhancement from OWL-S:** The MUSIC ontology allows to have a harmonized view on the QoS-properties of services. It establishes a common vocabulary of context information. Domain specific and general entities are divided. This is done by a two level hierarchy containing the top level concepts and the information concept. The top level provides general knowledge. For particular domains sub-ontologies are plugged in. These sub-ontologies are extensible and not fixed. The MUSIC Ontology uses OWL-S, as shown before in Section 3.4. It enhances OWL-S in some parts to support the semantic service discovery, description of service levels and service level agreements. The goal of this extension is to reach machine understandable information on QoS dimensions and on service requirements. Heterogeneity, mentioned as a requirement in the beginning of this section is reached via this semantic description.

The OWL-S description can be generated from the modeled component types and their provided and required ports and interfaces. This affects the input and output parameters as well as the datatypes. The resulting OWL-S description is taken and associated with the component type, this can then be used for service discovery. The QoS is defined in a contract between consumers and providers of a service. This is called Service Level Agreement (SLA). E.g. it describes the performance and the properties that are expected. OWL-S is extended by the specification of
QoS-dimensions, which are expected by the discovered service. This information is given in the specification of Service Level Dimensions (SLDs) and Service Levels (SLs). The SLDs are used in the property predictors and the utility functions. SLs are specifying the interest of consumers in QoS with regard to the SLDs. [12]

**Example:** In the following an example of a software is shown which can be modeled in this way with the MUSIC platform. This example follows the introduced example from Section 2. In order to show the most important modeling features, it is abstracted from the reality and focuses on self-adaptions.

The application that is going to be modeled is the same management application from the example. It supports services like accessing business and customer information, video calls and route planning. To reach its variability, the application can use several different implementations. They depend mainly on the available bandwidth and the power of the battery that is left. These variations are presented in the variability model as plans. The application can choose between five plans, as seen in Figure 5. These are NotOnlineClient, WeakOnlineClient, BasicOnlineClient, ExtendedOnlineClient and Standby.

In order to reach the variability one of the five realizations is chosen. The CompositePlans have to be resolved recursively and the BluePrint plans represent an end of the recursion.

The specification of a CompositePlan is done via further plans. An example of the component composition of the ExtendedInternetClient plan is shown in Figure 6. It consist of the component types UI which represents the user interface, VideoController which manages the video calls and normal calls for the application and BusinessController which manages all business process of the application.

An Example of the BluePrint plan of the VideoController is presented in Figure

![Fig. 5. Plans for the management application](image-url)
7. Here a utility function, ultimate component and some QoS properties for the ports of the component are given.

The QoS properties are given as expression regarding the bandwidth and the power level of the battery. The expressions have to follow some architectural limits. For example a mobile phone without UMTS cannot reach certain speeds of bandwidth.

**Enhancing MUSIC to reach support for SOAs:** The MUSIC platform has some extensions to reach support for service oriented architectures (SOAs)\(^4\). For this MUSIC introduces a new component, the SOA support component as composite component.

The service discovery is needed for publishing and discovering services using existing discovery protocols. With the remoting service, services are exported from the server side. This opens the possibility to accept requests from the consumer side, to bind them. A service description defines the provided functionalities, these are published by the service discovery. For a service discovery a discovery listener can be registered. This listener is interested in special services, this means that policies can exist to choose special services. For the discovered services plans are created so that the adaption manager can use the services. This also works vice versa when the service disappears.

\(^4\) Service-oriented architecture (SOA) are design principles used during systems development, functionality is represented by a suite of interoperable services [15].
Services can be offered in a set of service levels. When it is detected it has only an abstract description. For the needed service level, the complete description is provided, via an inheritance of the abstract description plus further data. This service negotiation is done during the service discovery. Because service negotiation is a time critical factor, this has to be completed as soon as possible. If a remote service is used later on, a component is specified that is acting as a service proxy. This encapsulates the communication which is used to access the service. It is possible to monitor the service via monitoring functions of the proxy, to ensure to follow the SLA. [13]

5 Conclusion

This paper has introduced a modeling approach for service-based adaptions in a QoS driven planning framework. It supports the manner of the MDA approach in software development. The framework especially deals with the issues of self-adaptive mobile applications. These applications support component and service-based reconfigurations. It was shown that the framework is able to react on changes of the environment, without lacking in the QoS. This happens because the framework always uses the adaption with the best overall utility for the application. It is described how to use semantic descriptions for the specification of variation points. This opens the possibility to find services dynamically, during the adaptation process and also to include this into the application. A main advantage of the solution, is the bridge of the gap between parameters needed for the adaptation in the planning framework and the descriptions of the services that will be discovered. This allows a view on the QoS properties of dynamically discoverable services and of properties of the applications. All this is reached by an enhancement of the OWL-S service description. The ontology is able to specify QoS and context properties. Based on this plus an UML based description for ports and interfaces, service proxies can be generated automatically. The service proxy realizes the integration of the service into the actual configuration. This description allows a heterogeneous description of the components.

The requirements were linked to the parts of the project that solve these. But even if all requirements are fulfilled, some problems are left. To specify all the plans for the variability model when only small changes are done which could be adjusted via parameters of the configuration leads to a big modeling effort. This could be changed, so that plans can be reused. The whole concepts are implemented and tested in the MUSIC approach. Different implementations enable the whole process from modeling to generated code. All the discussed concepts should be able to solve the issues of the example motivation introduced in Section 2. The solution of describing webservices their interfaces and the communication
with different webservices has to be specified in a more detailed way. This has to be done in the future work and is not sufficient in the current approach.
References