Model-Driven Architecture for Mobile Applications*

Jürgen Dunkel and Ralf Bruns

Hannover University of Applied Sciences and Arts, Department of Computer Science,
Ricklinger Stadtweg 120, 30459 Hannover, Germany
{dunkel,bruns}@fh-hannover.de

Abstract. Although significant improvements in the development of business applications for mobile devices have been made in recent years, the software development in this area is still not as mature as it is for desktop computers. Therefore, declarative and code generation approaches should be preferred instead of manually coding. In the BAMOS project an architecture has been designed and implemented for the generic and flexible development of mobile applications. The architecture is based on the declarative description of the available services. In this paper we present a model-driven approach for generating almost the complete source code of mobile services. By applying model-driven development within the proposed approach, a new service can be conveniently modeled with a graphical modeling tool and the graphical models are then used to generate the corresponding XML descriptions of the mobile user interface and the workflow specification. In order to use such a service no specific source code has to be implemented on the mobile device.

Keywords: model-driven architecture (MDA), mobile applications, XForms, meta models, code generation.

1 Introduction

Nowadays mobile devices, e.g. mobile phones, personal digital assistants (PDA) or smart phones, are ubiquitous and accompany theirs users almost every time and everywhere. Their capability of connecting to local area networks via Bluetooth or Wireless LAN potentially enables new types of mobile applications expanding the limits of present ones. So far, mobile devices do not fully exploit the whole potential of these networks. They are mostly employed only for communication or personal information management purposes.

While moving with a mobile device, the user enters a large number of different local networks; each might offer different localization-specific services. Examples for such location-based services [HaRo04] are the timetable and location plan of the next bus stop, the recent programs of the local cinemas, or a car reservation service of the car rental agencies nearby.

Today, the software development for mobile devices is cumbersome and not as mature as for desktop computers. Therefore, declarative and code generation approaches

* This work was supported by AGIP (Arbeitsgruppe Innovative Projekte beim Ministerium für Wissenschaft und Kultur des Landes Niedersachsen) under Research Grant F.A.-Nr.2004.612.

should be preferred instead of manually coding. In the BAMOS project (**B**ase **A**rchitecture for **MO**bile applications in **S**pontaneous networks) [SPBD05], an architecture has been designed and implemented for the flexible development of mobile applications. The BAMOS architecture can serve as a powerful base for code generation approaches. Using the BAMOS platform a mobile device can dynamically connect to a local network and use all the available services offered there. To make this approach successful the development of mobile services should be as easy as possible. In this paper we present a model-driven approach for generating nearly the complete source code of mobile BAMOS services. Furthermore, on the mobile devices no line of code has to be implemented when the BAMOS platform is used.

The paper is organized as follows: in section 2 we outline the architectural approach of the BAMOS platform which provides the destination platform for our model-driven development. Subsequently, section 3 motivates the usage of a model-driven architecture and derives a meta model for a domain specific language (DSL). An example is presented that illustrates the proposed approach. Finally, section 4 summaries the most significant features of the approach and provides some directions of future research.

2 Architectural Approach

An indispensable prerequisite for applying model-driven development is a powerful architectural base providing the destination platform for code generation. The BAMOS platform enables the development of mobile applications by providing two software components. The first component is an Adhoc Client that – similar to a Web browser – enables the mobile device to access information services in spontaneous networks. The second component is a Service Broker that – similar to a Web Server – serves as an interface between the Adhoc Client and the services available in the network.

With the BAMOS platform a mobile device can use different services in diverse local networks. The Adhoc Client is a generic software component that does not require any information about the specific services. It loads the declarative descriptions of the services at run-time and generates a service-specific graphical user interface. The core concept underlying this generic approach is the declarative description of the process flow as well as of the graphical user interface.

2.1 BAMOS Components

The BAMOS platform serves as the implementation base for the generation of mobile applications. It consists of three main components. Figure 1 illustrates the architecture and the relationship between the different architectural components.

The **Service Provider** offers services to other systems. To access these services on a mobile device some prerequisites have to be fulfilled:

– The implemented services must be accessible for remote programs. For example they may be implemented as a Web Service that can be invoked over the Internet.

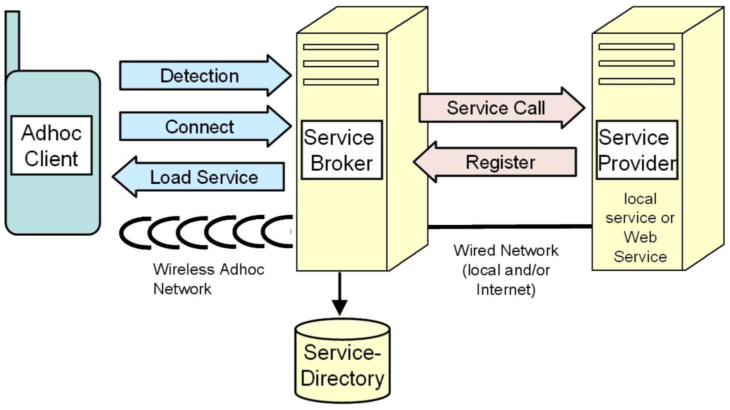


Fig. 1. BAMOS components

- In BAMOS all services must be described in a declarative manner to permit their usage on a mobile device. Each service description defines the mobile user interfaces and the corresponding control flow (more details are discussed in section 2.2.).
- The service providers have to register their service descriptions at the Service Broker.

The **Service Broker** mainly acts as a mediator between Service Providers and Adhoc Clients. It can be described by the following characteristics:

- It is integrated into two different networks: on the one hand in a local wireless network (e.g. Bluetooth) for connecting with the mobile devices, on the other hand in a wired network (local area network or Internet) for accessing the services provided by the Service Providers.
- It delegates the client service requests to the appropriate Service Provider and forwards the response to the Adhoc Client.
- It holds a service directory where all available services must have been registered. The directory contains the declarative descriptions of all available services. Services can be published and searched in the directory.

The **Adhoc Client** is a software component that is running on a mobile device. The device can enter and leave a local wireless network. In this case, the Adhoc Client acts as part of a wireless adhoc network. It provides the following features:

- When entering a local wireless network, it connects spontaneously to the Service Broker (using Bluetooth or WLAN).
- It loads the declarative service descriptions from the Service Broker for generating a user interface on the mobile device. Afterwards the user can enter data on the mobile device that is sent as a service request to the Service Broker. The Broker delegates the request to the Service Provider offering the requested service.

The presented BAMOS architecture allows an Adhoc Client to use different kinds of services, e.g. services that are generally available like Web Services. This architetural concept is independent of particular data transfer technologies in adhoc networks. The communication between Service Broker and Service Provider exploits common network technologies; this aspect is not further considered in this paper. The origin of the services is transparent to the Adhoc Client because the Service Broker is its only communication partner.

2.2 Service Descriptions

The description of a service must specify the mobile user interface of the service as well as the sequence of steps necessary in order to perform the complete service. To use a service on a mobile device normally a sequence of different screens is necessary: for selecting the desired service, for entering the input data and for presenting the output information returned by the service. The mobile user interface can be characterized by two different aspects:

(a) The layout of each screen on the mobile device.
(b) The workflow determining the sequence of screens on the mobile device.

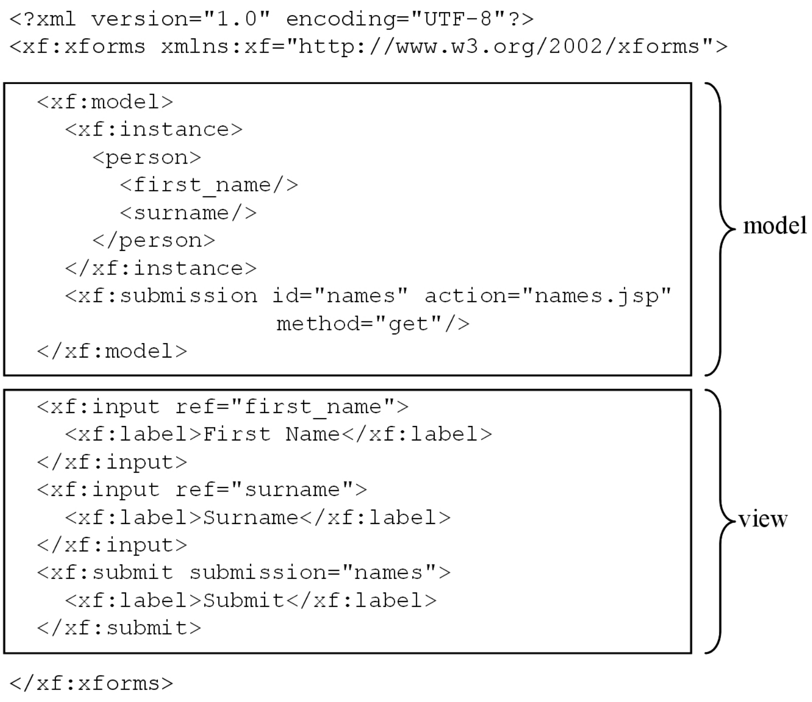
(a) Specification of Mobile User Interfaces by XForms
Although the Adhoc Client is domain-independent, it should be able to interact with domain-specific services. Thus, in order to cooperate with such a service the client requires a description of the mobile user interface. This user interface description is provided by the Service Broker and can be accessed by the client. With XForms a W3C standard has been chosen as the mobile user interface description language. The main advantage of XForms is its close correlation to MIDP, the core technology used to implement graphical user interfaces on mobile clients. MIDP has been chosen as the implementation technology for the Adhoc Client.

The original intention of XForms was to build the next generation of forms in the World Wide Web. XForms is a XML-based language, issued as an open standard by W3C, with several improvements compared to traditional HTML forms [XFor06].

MIDP (Mobile Information Device Profile) is a J2ME profile suitable for the development of simple, but structured mobile user interfaces [Sun06]. MIDP user interfaces show significant similarities to traditional HTML forms: elements like input fields, radio buttons or lists offer very similar input capabilities. In addition, the possibilities for human-computer interaction are very often restricted to the submission of the entered input data – also similar to HTML forms. Thus, MIDP elements can be directly mapped to XForms and, consequently, XForms has been chosen for describing the mobile user interface.

Structure of XForms
One of the main concepts of XForms is the clear separation of model and view. An XForms document consists of two parts: The model part contains the data of the form, which can be displayed and altered. In the example shown in figure 2, the model contains the first name and surname of a person. The submission element holds the information about the action that shall be executed on the model data. In addition to the model part, the second part of an XForms document specifies the visual presentation of the model data as well as the possibilities for user interaction. Similar to HTML forms, XForms offers several input and output elements. Every presentation element

refers to a model element by means of a ref attribute. In the example, two input fields enable the presentation and alteration of the first name and surname of the person. The submit element specifies the presentation component for presenting the submission specified in the model, typically a button. If the submit button is pressed, all input fields are mapped to the elements in the model and the model is processed according to the action specified in the submission element. In the context of the WWW, the browser would transfer the model to a web server. In the context of BAMOS, the model is transferred to the Service Broker, e.g. via Bluetooth.



```
<?xml version="1.0" encoding="UTF-8"?>
<xf:xforms xmlns:xf="http://www.w3.org/2002/xforms">

  <xf:model>
    <xf:instance>
      <person>
        <first_name/>
        <surname/>
      </person>
    </xf:instance>
    <xf:submission id="names" action="names.jsp"
                   method="get"/>
  </xf:model>

  <xf:input ref="first_name">
    <xf:label>First Name</xf:label>
  </xf:input>
  <xf:input ref="surname">
    <xf:label>Surname</xf:label>
  </xf:input>
  <xf:submit submission="names">
    <xf:label>Submit</xf:label>
  </xf:submit>

</xf:xforms>
```

Fig. 2. XForms example

Figure 3 displays the message exchanges between the Adhoc Client on the mobile device and the Service Broker. XForms serves as the message format in every BAMOS interaction. (1) The Service Broker holds the XForms description of the mobile user interface and instantiates a specific XForms document for the requested interaction. (2) This XForms document is transferred via a wireless network to the mobile device where (3) the Adhoc Client renders the mobile user interface according to the information in the received XForms. (4) After an XForms submit the corresponding XForms model data is transferred back to the Service Broker where it is processed, usually by invoking an offered service. (5) The XForms for the subsequent interaction step is determined, an instance of it is generated and the output data of the service call is included in it. (6) This new XForms document is send to the mobile device and the interaction process can proceed.

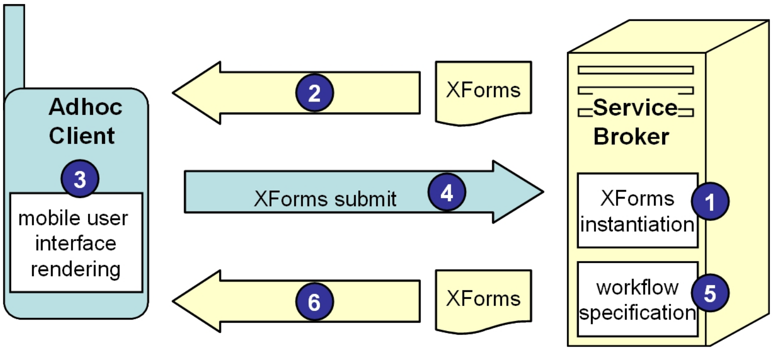


Fig. 3. Message exchanges between Adhoc Client and Service Broker

(b) Workflow Specification

In addition to the declarative description of the mobile user interface by XForms, the Service Broker has to determine the sequence of the dialogue steps necessary to execute a service, i.e. which XForms document has to be displayed next in response to a submit, and to invoke the requested service operation on the Service Provider. The core concept of the Service Broker is a process control component that can interpret the declarative service descriptions stored in the service registry.

In order to describe the process flow a simple XML-based workflow language has been designed. The concept of this language is based on the concept of a service in BAMOS and its parts. Figure 4 illustrates in detail the components of a service in BAMOS.

A BAMOS service consists of the sequence of interactions, which must all be performed in a predefined order to complete the service. A service can be a very simple one with a limited scope, e.g. querying information. Yet, it can also be a complex, composite service that is implemented by invoking other services, e.g. weather forecast service and timetable service for the public transport.

Every interaction step is implemented by a screen (a XForms template) displayed to the user on the mobile device. This template usually contains information reflecting the actual processing status of the previous interactions. The user can enter or alter data in the template and can issue a submission. Through a submission the input data is transferred as parameter to the service operation that is invoked. The output of this service operation serves as the initial input for the subsequent interaction, i.e. the next

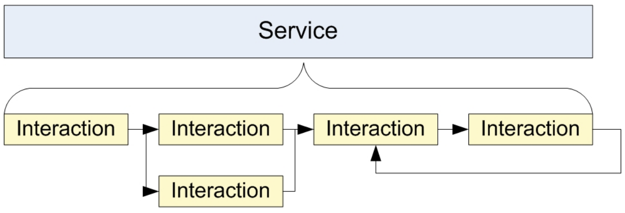


Fig. 4. Service composition and interactions

screen. Within the current interaction the next interaction step has to be determined. Due to the user input different conditional interactions can follow a finished one.

The presented concept of a BAMOS service is modeled by an XML workflow language. Figure 5 shows an example of the XML description of the interactions constituting a service.

```
<service id="MyService" caption="MyService">
  <class name="de.fhhannover.impl.MyService"/>

  <!-- Step1 -->
  <interaction id="Step1" start="yes">
    <xform src="Step1.xml"></xform>
    <method name="performStep1" />
    <next decisionpath="/MyService/choice">
      <case content="weatherService"   target="Step2"/>
      <case content="transportService" target="Step3"/>
    </next>
  </interaction>

  <!-- Step2 -->
  <interaction id="Step2">
    <xform src="Step2.xml"></xform>
    <method name="getStep2">
      <param id="0" path="/Step2/input"/>
      <outparam name="result" id="0"/>
    </method>
    <nextdefault target="Step3"/>
  </interaction>
  ...
</service>
```

Fig. 5. Service interactions specified in XML workflow language

The element service is the root element and contains several interaction elements. Every interaction element contains information about the corresponding XForms template (in the xform element) and about the service operation to be called with the mapping of the input and output attributes (specified in the element method).

The elements next and nextdefault define the subsequent interaction and determine the sequence of the interactions constituting a service.

3 Model-Driven Development

As discussed above, an XML-based specification of the mobile user interface and the workflow description enables the development of mobile applications in a very generic and flexible manner. In order to make a new service available in BAMOS, the descriptions of its mobile user interface as well as its workflow specification have to be defined in XML format. Only the service itself has to be implemented by the Service Provider in any programming language and wrapped by a Web Service.

To make the BAMOS concept successful the development of services should be as easy as possible. The entire effort for implementing services is already moderate but writing XForms screen descriptions and especially XML workflow descriptions for a service is too error-prone and time consuming. XML code is intended for the usage of software programs and cumbersome for humans as the short examples in the previous section illustrate. Model-driven development [Schm06], [AtKu03], [SVBH06] provides an approach to cope with these problems. It is based on the systematical use of models as primary artifacts in the software engineering process. By applying model-driven development within the BAMOS approach, a new service can be conveniently modeled with a graphical modeling tool and the graphical models are then used to generate the corresponding XML descriptions of the mobile user interface and the workflow specification.

The prerequisite for code generation is a semantically rich model. Because general-purpose modeling languages like UML [UML03] do not provide enough information for code generation a Domain Specific Language (DSL) is required that contains the necessary details for the automatic code generation. A DSL describes the elements of a certain modeling language and therefore can be considered as a meta model. Transformation engines and generators allow to analyze DSL models and to create artifacts such as other models, source code or XML deployment descriptors.

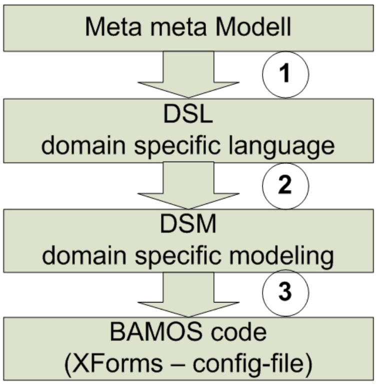


Fig. 6. Model-Driven Development steps

Figure 6 outlines the approach. First the DSL (or meta model) must be formally defined. This can be achieved by means of a model describing a meta model, i.e. a meta metamodel (step 1). Then the DSL can be used for domain-specific modeling (DSM); in this stage the mobile services are being modeled (step 2). Then the DSM model is finally used for generating the code; in this stage the XForms and the workflow description of the BAMOS platform are created (step 3). The four levels of description correspond to the four meta levels of the OMG Model-Driven Architecture: M0 - data, M1 - models, M2 - metamodels, M3 - meta metamodels [MOF04]. Each step is described in some more detail in the following subsections.

3.1 Specifying a Domain Specific Language (DSL)

General-purpose modeling languages for designing, specifying and visualizing software systems are not sufficient for code generation. They lack domain-specific model elements and concepts that specify the details required for generating the code. In our case the domain is about mobile applications based on the BAMOS framework. For example, a domain-specific model element could specify that a class attribute should appear as a choice box on a mobile device screen.

To let the code generators make use of the domain-specific model elements they must be defined in a consistent and formal way. Modeling languages can be formally defined by meta-modeling languages as Meta Object Facility (MOF) of OMG [MOF04] or Eclipse Encore [BSME03]. In a meta-modeling language the key concepts in a domain, their corresponding relationships, semantics and constraints can be precisely specified.

Domain-specific languages are mostly based on extensions of the Unified Modeling Language (UML) the de-facto standard for modeling languages. A pragmatic approach for defining a DSL are UML profiles [UML03] which use the built-in extensibility mechanisms of UML: stereotypes, tagged values and constraints. Figure 7 shows the meta model of a DSL for mobile applications based on BAMOS. The introduced new modeling elements inherit from the MOF-concepts MOF::Class and MOF::Attribute.

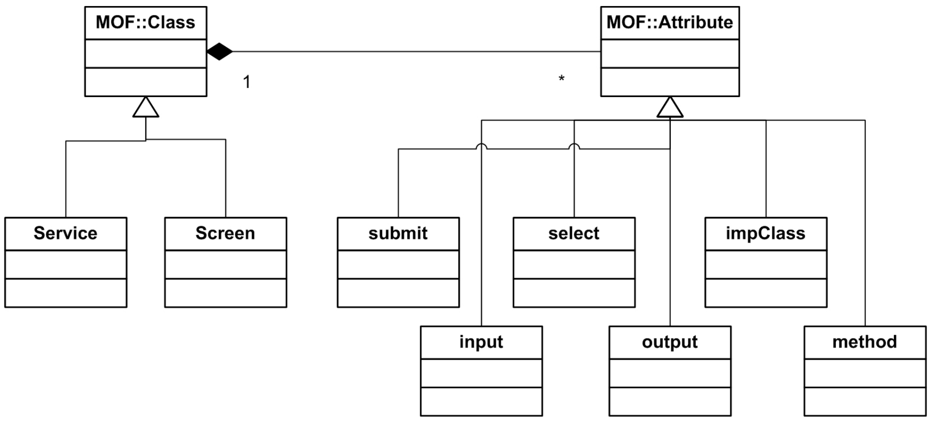


Fig. 7. The meta model of the BAMOS DSL (in parts)

The DSL defines two meta classes (inheriting from MOF::Class): a Service class describes mobile services and a Screen class describes screens on a mobile device. Furthermore, different types of attributes are defined for these classes (inheriting from MOF::Attribute). Special meta attributes for the Screen classes are used to define elements of the graphical mobile user interface: e.g. input defines an input field, select a radio button and submit a command button. For the Service class the implClass and the method attributes define the name of the method and the class of the Service

Broker which delegates a service invocation. The precise dependencies between the new model elements are defined by OCL constraints.

The sequence of interactions, i.e. the workflow between different mobile device screens can be specified by UML activity diagrams. Depending on the user input the Service Broker selects different interactions with corresponding XForms screens. Activity diagram guards are used for annotating transitions and specifying the appropriate interaction. An example is given in figure 8: depending on the value of attribute selectedService in screen1 the control flow will be directed to screen2 or to screen3.

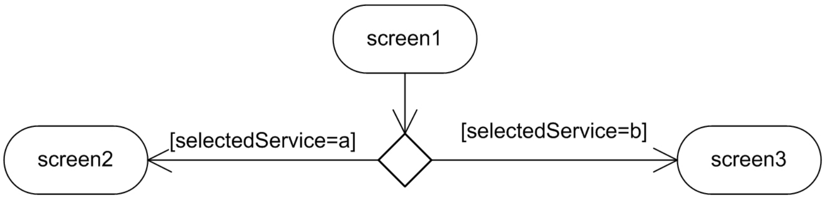
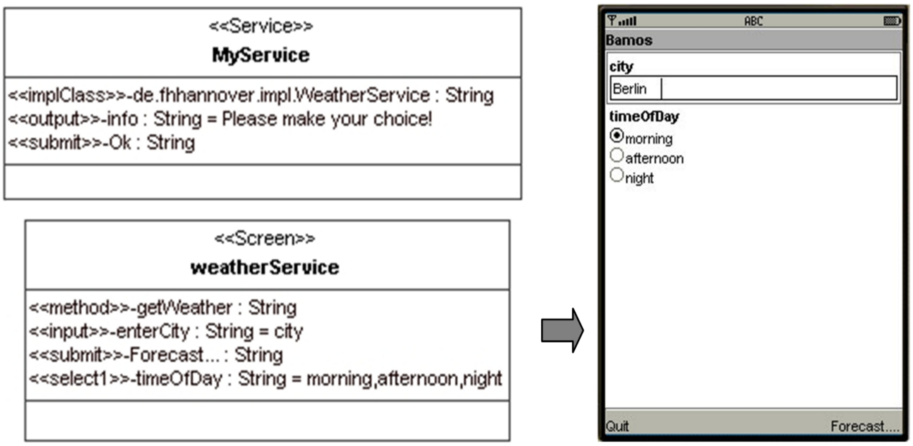


Fig. 8. UML activity diagrams for specifying the control flow

3.2 Domain-Specific Modeling (DSM)

Mobile services can be easily modeled using the DSL defined in the previous subsection. First the screens on the mobile device and the corresponding services must be specified. This can be achieved by a class diagram using stereotypes defined by the BAMOS DSL of figure 7. An example is presented in figure 9: the Service class MyService defines a service, which contains all the methods that are needed in the interactions of MyService. The Screen class weatherService defines an XForms screen,



```
<<Service>>
MyService
<<implClass>>-de.fhhannover.impl.WeatherService : String
<<output>>-info : String = Please make your choice!
<<submit>>-Ok : String
```

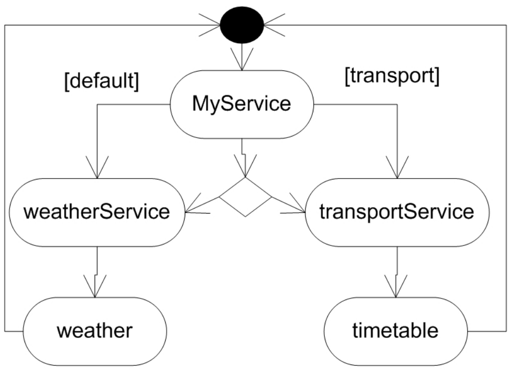
```
<<Screen>>
weatherService
<<method>>-getWeather : String
<<input>>-enterCity : String = city
<<submit>>-Forecast... : String
<<select1>>-timeOfDay : String = morning,afternoon,night
```

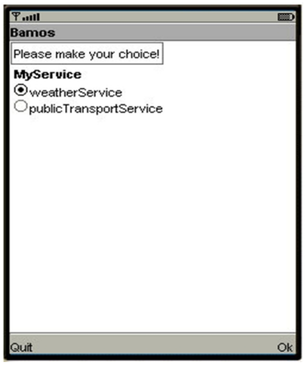
Bamos
city
Berlin
timeOfDay
[✓ morning]
afternoon
night
Quit Forecast....

Fig. 9. Service and Screen class in the DSM

which is presented on a mobile phone as illustrated in Figure 9 to let the user enter a city and a time for the weather forecast. The attribute enterCity with stereotype <<input>> defines an input field to specify the city a weather forecast is requested for. The attribute timeOfDay with stereotype <<select1>> is presented as a radio button for choosing one of the initial attribute values (morning, afternoon, night).

In a second step, the control flow must be defined by an activity diagram as shown in figure 10. Depending on the choice made in the mobile user interface an appropriate sequence of screens is sent to the mobile device. The right side of figure 10 shows the corresponding initial screen to choose one of the two available services.





Please make your choice!

MyService

● weatherService
○ publicTransportService

Fig. 10. Activity diagram specifying the control flow

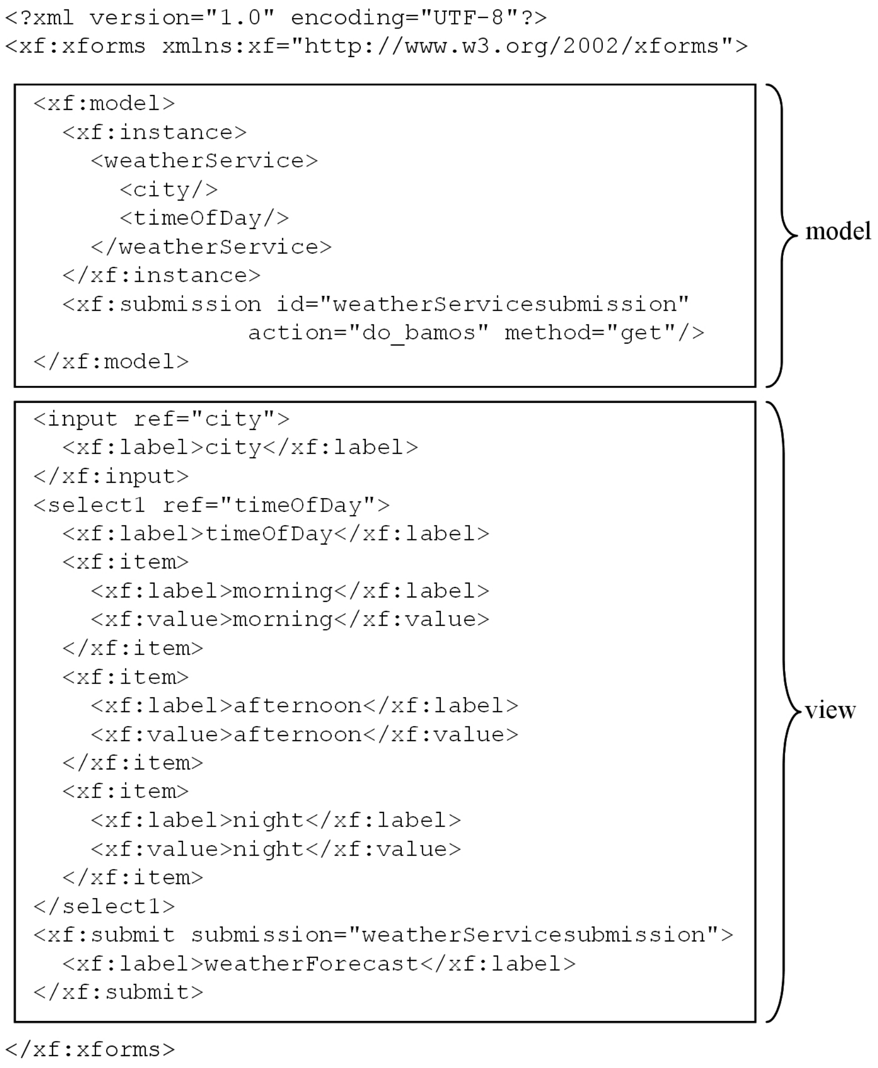
3.3 Code Generation

In a final code generating step the DSM model must be transformed into code artifacts. Transformation engines and generators first analyze a certain DSM model and then synthesize various types of artifacts, such as source code, simulation inputs, XML deployment descriptions, or alternative model representations. To make DSM models processable by code generators the OMG standard interchange format XMI (XML Metadata Interchange) [XMI06] can be used. A XMI model representation can be imported by transformation engines. Each engine usually provides its own proprietary transformation languages for code generation, e.g. Java ServerPages, XPand. Currently the OMG is working on a standard called QVT.

In our domain the XForms and the workflow specification documents of the BAMOS platform are generated. Figure 11 shows the XForms code of the weatherService screen according to the model described in figure 9.

The corresponding part of the generated workflow description document is presented in figure 12. It contains the definition of the MyService service specifying its implementation class MyService and a corresponding XForms file MyService.xml. The next decisionpath element determines the sequence of interactions: if a user selects weatherService in the radio button choice of the MyService XForms screen, the cotrol flow is directed to the weatherService interaction, otherwise the publicTransportService interaction is chosen.

The weatherService interaction refers to the file weatherService.xml containing the XForms code of figure 11 and to the method getWeather of the Service class MyService. The two parameters of the method (id=0 and id=1) correspond to the values of the elements in the XForms screen (i.e. input fields city and time, see figure 11 and figure 9).



```
<?xml version="1.0" encoding="UTF-8"?>
<xf:xforms xmlns:xf="http://www.w3.org/2002/xforms">

 <xf:model>
   <xf:instance>
     <weatherService>
       <city/>
       <timeOfDay/>
     </weatherService>
   </xf:instance>
   <xf:submission id="weatherServicesubmission"
                action="do_bamos" method="get"/>
 </xf:model>

 <input ref="city">
   <xf:label>city</xf:label>
 </xf:input>
 <select1 ref="timeOfDay">
   <xf:label>timeOfDay</xf:label>
   <xf:item>
     <xf:label>morning</xf:label>
     <xf:value>morning</xf:value>
   </xf:item>
   <xf:item>
     <xf:label>afternoon</xf:label>
     <xf:value>afternoon</xf:value>
   </xf:item>
   <xf:item>
     <xf:label>night</xf:label>
     <xf:value>night</xf:value>
   </xf:item>
 </select1>
 <xf:submit submission="weatherServicesubmission">
   <xf:label>weatherForecast</xf:label>
 </xf:submit>

</xf:xforms>
```

Fig. 11. Example of the generated XForms code for the weatherService screen

```
<!--MyService-->
<service id="MyService" caption="MyService">
  <class name="de.fhhannover.impl.MyService"/>

  <interaction id="MyService" start="yes">
    <xform src="MyService.xml"></xform>
    <nextdefault target="weatherService"/>
    <next decisionpath="/MyService/choice">
      <case content="weatherService" target="weatherService"/>
      <case content="publicTransportService"
                               target="publicTransportService"/>
    </next>
  </interaction>

  <!-- weatherService -->
  <interaction id="weatherService">
    <xform src="weatherService.xml"></xform>
    <method name="getWeather">
      <param id="0" path="/weatherService/city"/>
      <param id="1" path="/weatherService/time"/>
      <outparam name="result" id="0"/>
    </method>
    <nextdefault target="weatherResult"/>
  </interaction>

  <!--publicTransportService-->
  <interaction id="publicTransportService">
    <xform src="publicTransportService.xml">
  ...

</service>
```

Fig. 12. Part of the generated workflow description file

4 Conclusion

In this paper, we have described a model-driven approach to generate applications for mobile devices. Model-driven architecture (MDA) provides a higher level of abstrtion for developing software: it allows modeling software systems instead of programming. Only a few domain-specific functionalities remain for manual implementation. The indispensable prerequisite for an MDA approach is a powerful architectural base providing the destination platform for code generation and the development of a Domain Specific Language (DSL).

We introduced the BAMOS platform which allows the specification of complex mobile application using XML files to generate XForms screens for mobile devices.

The presented model-driven approach avoids the error-prone coding of XML files. Altogether, MDA shortens significantly the development time and improves software quality. Because UML models are the main artifacts instead of XML code, maintenance and reuse of model elements is made easier.

One drawback of the approach is that many different tools must be used and integrated. Actually, most UML modeling tools do not satisfactorily support meta modeling and the code generating tools are still proprietary and not yet stable.

References

[AtKu03] Atkinson, C., Kuhne, T., Model-driven development: a metamodelling foundation, IEEE Software, IEEE Computer Society, vol. 20, pp. 36-41, 2003.

[BSME03] Budinsky, F., Steinberg, D., Merks, E., Ellersick, R., Grose, T. J., Eclipse Modeling Framework: A Developer's Guide: Addison Wesley, 2003.

[DeKV00] Deursen, A. V., Klint, P., Visser, J., "Domain-specific languages: An annotated bibliography," ACM SIGPLAN Notices, vol. 35, pp. 26-36, 2000.

[HaRo04] Hadig, T.,Roth, J.: Accessing Location and Proximity Information in a Decentralized Environment, International Conference on E-Business und Telecommunication Networks, Setúbal, Portugal, 2004, pp. 88-95

[Herr03] Herrington Jack. Code generation in action. Manning Ed. 2003.

[MDA06] OMG. MDA. http://www.omg.org/mda

[Sun06] Sun: Mobile Information Device Profile MIDP) http://java.sun.com/products/midp/

[MOF04] MOF, "Meta Object Facility 2.0 Core Specification" 2004, DocId: ptc/03-10-04.

[oAW06] openArchitectueWare: http://www.openarchitectureware.org/

[Schm06] Schmidt, D.C. (February 2006). Model-Driven Engineering. IEEE Computer 39 (2). Retrieved on 2006-05-16.

[SVBH06] Stahl, T, Völter, M., Bettin, J., Haase, A., Helsen S., Model-Driven Software Development: Technology, Engineering, Management: Wiley, 2006.

[SPBD05] Schmiedel, M., Pawlowski, O., Bruns, R., Dunkel, J., Nitze, F., Mobile Services in Adhoc Networks, in: Proc. of Net.ObjectDays 2005, Erfurt, Germany, 2005, pp. 167-178.

[UML03] UML2.0, "UML 2.0 Superstructure Specification, Final Adopted Specification, available at www.omg.org," 2003

[Wile01] Wile, D. S., Supporting the DSL Spectrum, Journal of Computing and Information Technology, vol. 9, pp. 263-287, 2001.

[XFor06] W3C, The Forms Working Group: http://www.w3.org/MarkUp/Forms/

[XMI06] OMG/XMI XML Model Interchange (XMI) 2.0. Adopted Specification. Formal/03-05-02, 2003.