Information and Software Technology 51 (2009) 1244–1260

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Model-driven development of composite context-aware web applications

Georgia M. Kapitsaki [a,*], Dimitrios A. Kateros [a], George N. Prezerakos [b,1], Iakovos S. Venieris [a]

[a] *Intelligent Communications and Broadband Networks Laboratory, School of Electrical and Computer Engineering, National Technical University of Athens, 9 Heroon Polytechneioy Str., Zografou, 15780 Athens, Greece*

[b] *Software and Service Engineering Laboratory, Technological Education Institute of Piraeus, Dept. of Electronic Computing Systems, Petrou Ralli & Thivon 250, 12244 Athens, Greece*

A R T I C L E I N F O

Article history:
Received 27 May 2008
Received in revised form 27 January 2009
Accepted 10 March 2009
Available online 27 March 2009

Keywords:
Context
Context-awareness
Model-Driven Architecture
Web services

A B S T R A C T

Context-awareness constitutes an essential aspect of services, especially when interaction with end-users is involved. In this paper a solution for the context-aware development of web applications consisting of web services is presented. The methodology proposes a model based approach and advocates in favour of a complete separation of the web application functionality from the context adaptation at all development phases (analysis, design, implementation). In essence, context adaptation takes place on top of and is transparent to the web application business functionality. Starting from UML diagrams of independent web services and respective UML context models, our approach can produce a functional composite context-aware application. At execution level this independence is maintained through an adaptation framework based on message interception.

1. Introduction

One of the most desired features of an end-user service, from an operator's as well as from a developer's viewpoint, is the possibility to easily adapt the service to the preferences and needs of each individual end-user. This process, known as contextual adaptation, requires the service to exploit past and present information about the user such as current situation, location, profile, etc. The respective services are often characterized as context-aware services (CAS). Context can be seen as a set of information that includes user's activity, location, personal preferences and current status, while the most widely accepted formal definition has been provided by Dey and Abowd [6]: "*Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves*".

At the same time, there is an increasing trend towards offering Internet services to mobile users, as the advances in mobile devices capabilities allow for the provision of stripped down versions of traditional web applications. The provision of context-aware services is especially sought after in the area of mobile computing since the usage of mobile devices poses certain constraints [2], such as minimal user to service interaction and limited keyboard and screen size. Therefore the use of context information can play an important part as mobile service providers can make substantial use of this information, in order to provide new services (e.g. location based) or enhance the user experience of existing services.

The following three observations regarding Internet service provision can be particularly useful when it comes to the provision of such services to mobile users:

1. As the number of available web services (WSs) increases, the development of web applications that integrate existing web services has become more and more popular. In this manner, WSs are perceived as autonomous building blocks usable for the composition of richer Internet services. The majority of research activities regarding web service composition focuses on business to business interactions [10]; however, there is recently relevant activity regarding the creation of user-centric web applications. For example, mashups, which aggregate content and functionality from various sources providing new innovative services, typically utilize external web services as data sources.
2. Any user-oriented web application can be decomposed to a number of steps, each one concerned with a specific portion of user–application interaction. Furthermore, the wide adoption of the Model-View-Controller (MVC) pattern that is considered to be the standard for web application development provides sufficient decomposition of the application into a number of components whose roles are specialized [24]. MVC frameworks (e.g. Apache Struts[2]) further facilitate development by allowing a

* Corresponding author. Tel.: +30 210 772 2551; fax: +30 210 772 1092.
E-mail addresses: gkapi@icbnet.ntua.gr, georgina.kapi@gmail.com (G.M. Kapitsaki), dimitris@icbnet.ntua.gr (D.A. Kateros), prezerak@teipir.gr (G.N. Prezerakos), venieris@cs.ntua.gr (I.S. Venieris).
[1] Tel.: +30 210 538 1132; fax: +30 210 538 1260.

[2] <http://struts.apache.org/>.

more straightforward mapping of the user interface (UI) flow to the contents of the source and configuration files. Apparently, business objects of the application model can be simply consumers of existing web services. Effectively, the web application, at design level, can consist of a diagram involving a set of states, actions executed on the states and events that trigger transitions between states, where the executed actions involve interaction with existing web services.
3. As far as the context adaptation of such services is concerned, it can be noted that although the interaction between the end-user and the service can be adapted to contextual parameters, the overall goals related to service logic are usually indifferent to context changes. For example, consider a service providing airport ticket bookings. Depending on the end-user context (e.g. financial constraints, preferred airlines) different operations belonging possibly to different subservices can be invoked. However, in every case the service goal remains the same regardless of the specific end-user context. Along this line of thought, core service logic and context handling can be treated as two separate concerns, both on modeling and implementation level.

The previous observations depict the potential for the development of context-aware web applications intended for mobile users based on existing web services. This approach has several advantages, most notably:

- Existing business layer components can be exposed as web services and be reused; in this manner code reuse is promoted and application development is based on proven and tested software modules.
- Third party web services can be utilized, thus reducing the development effort and integrating useful services into new, attractive applications.
- The contextual adaptation is performed while using context-agnostic components and can be modified without needing to intervene in their implementation.

In this paper, a methodology for the development of such applications is proposed. This methodology adopts a Model-Driven Engineering approach that employs Unified Modeling Language (UML) diagrams [18] during the design phase. It is shown that the concrete UML model is sufficient to result in the automatic generation of a functional web application consisting of web services. The modeling exploits a number of pre-defined profiles, whereas the target implementation is based on an architecture that performs context adaptation of web services based on interception of Simple Object Access Protocol (SOAP) messages. Furthermore, it is shown that the context adaptation process is entirely transparent to the core web application.

The following section presents related research efforts that inspired the main ideas contained in this paper.

2. Initial motivation and related work

One way to achieve the separation of business logic from context-dependent behaviour is to have the service in question composed of a service logic skeleton responsible for the main business goals of the service combined with several code stubs that would handle the context-dependent functionality. This idea of a skeleton and stub approach originates from the work carried out under the general term of Context-Oriented Programming (COP), which constitutes one of the first significant attempts of context adaptation in the implementation level. COP [13] makes use of open terms in program logic, goals that describe an entity's role, contexts and stubs to extract context information from the main execution skeleton. Context-filling is done by matching and binding the open terms to external code parts called stubs which provide the desired adaptation. The Python programming language was used in COPs implementation, whereas stubs for context filling are stored in a stub repository.

In the field of web services, message interception has been suggested as a mechanism that separates business logic from context adaptation. Service requests and responses travel between web service clients and web services encapsulated in envelopes following the SOAP format. Context information related to a specific service can be handled by (i) extending the syntax of SOAP messages in a way that allows the inclusion of context information and (ii) intercepting these messages on the way and processing the embedded context information, leaving the context handling, as much as possible, outside web services themselves [14]. Processing can be performed either on the client or on the web service side using a specific Context API. This approach provides for substantial flexibility in handling context processing independently from the core service logic, although the manipulation of metadata on SOAP headers produces an undesired binding of the client software with the context adaptation mechanism.

At the same time, as web application development has evolved, a number of model-driven approaches have been proposed for the simplification and automation of the development process. A high level categorization of these can be performed taking into account whether they employ a Domain Specific Language or UML, which is a general purpose modeling language.

Two of the most notable web modeling methodologies and notations are WebML [4] and OO-HDM [21]. The WebML project has defined four models: the structural model (data content, entities and relation), the Hypertext Model (composition and navigation), the Presentation Model (layout and graphic appearance of page) and the personalization model (individual content based on users and groups). A CASE tool, called WebRatio, supports the software development lifecycle based on the WebML methodology. OO-HDM is an object-oriented method which focuses on the authoring process and is based on various models describing information (conceptual), navigation and interface aspects of these applications. A code generation process takes place exploiting the mapping of these models into running applications, in various environments. HyperDE is an open source environment supporting OOHDM built upon Ruby on Rails[3] MVC framework. An interesting observation on the aforementioned methodologies is that they focus on information modeling rather than modeling the functional design of the web application.

Another set of tools aiming at the automation of web application development employ a more open approach which includes purely UML notation (standard UML as well as UML profiles). These are known as Model-Driven Architecture (MDA) [16] generators (e.g. AndroMDA[4] and Acceleo[5]) and focus on the generation of a significant portion of the application's source code. Their functionality relies usually on parsing the XML Metadata Interchange (XMI) [19] representation of the defined UML models and applying a number of templates, in order to generate the target source code. Optionally, the developer will have to add or edit source code portions. Then, integrating with build and project management tools they can produce and deploy the final web application. Work in this direction has been performed in [12] where the automation of the generation of a web application designed around a user–service interaction flow based on UML diagrams is discussed. An obvious advantage of this approach lies in the adoption of open standards. The modeling is suf-

[3] <http://www.rubyonrails.org/>.
[4] <www.andromda.org>.
[5] <www.acceleo.org/>.

ficiently decoupled from the code generation so that the user can employ a number of different UML modeling tools. Furthermore, the migration to new web application platforms and the utilization of innovative web application frameworks can be performed independently, as it is a matter of model tweaking and specification of appropriate code generation templates.

Model-driven techniques have also been investigated in the field of context-aware service development. One of the first approaches towards modeling the interaction between context information and service is found in ContextUML [22]. ContextUML is a UML meta-model, which extends the existing UML syntax by introducing appropriate artifacts, in order to enable the creation of context-aware service models. ContextUML-derived models consist of class diagrams where classes correspond to context as well as service constructs, while UML dependency and association relationships express the interaction between context and service. The work in ContextUML has been extended in [20] towards the direction of further decoupling context model from service model design. Apart from class diagrams, UML activity diagrams were also used in this attempt for modeling the core service logic flow along with MDA techniques and AOP. Another UML profile towards model-driven development of context-aware applications similar to ContextUML can be found in [9]. In this profile, context is categorized in state-based context that characterizes the current situation of an entity and event-based context that represents changes in an entity's state. Constraints are used on both context types to trigger various invocations: state constraints refer to specific points in time, while event constraints exploit historical data of context events. Context information can be either used to be mapped to specific values or modify the structure or the behaviour of the application. This UML model is then transformed to AspectJ[6] code.

Finally, a framework supporting the design of context-aware multichannel web applications has been proposed in [5]. Context is rendered as a first-class actor allowing applications to adapt automatically to context changes without user intervention. The design of data (application data) is separated from the design of hypertext (application front-end). That way application data is enhanced with context information and then the hypertext is adapted to this context. Context information is added to the application data in the form of metadata, whereas context-aware capabilities are added in hypertext by adding context-aware pages (C-pages) responsible for interactions for context information retrieval. In the implementation level, context adaptation is performed mainly on the web application controller causing a significant binding of the web application programming with the context adaptation logic.

Starting from the main ideas presented above, the aim of this paper is to present a model-driven methodology for the creation of context-aware composite services targeting mobile clients out of existing services belonging to third parties or wrapped around existing legacy systems (which, in both cases, cannot be easily modified) using, as much as possible, off-the-shelf software tools. For these reasons context-awareness is handled at the service interface level that consists of operation invocations and the exchange of respective input/output parameters. In this manner the adaptation mechanism is transparent to the end-user service programming and alterations to the context adaptation policies can be performed without affecting the service functionality. In order to maintain a terminal independent implementation as much as possible, the application is designed as a web application. The modeling is performed in UML notation, in order to maintain a clear separation between the modeling tools and the application generation framework. Specifically, the service flow is modeled around a state transition diagram and separate models are used to model the external web services interfaces.

In comparison to the above mentioned approaches context management is not handled only on design or implementation level but is integrated into all development stages (analysis, design and implementation). At the same time the base application functionality is independent from the context adaptation logic, which is executed separately. Furthermore, context data are not incorporated into the main application, but maintained independently. Focus is given on the service functional design and application flow that indicates the order in which services are invoked and the actions that trigger these invocations. This way the web application design is not based on the service context adaptation and the application controller is left intact. Lastly, the use of UML instead of a domain specific language allows the use of already available modeling environments in order to visualize and edit the models by a wider audience. In the following section the proposed methodology is outlined along with the steps and technologies involved.

3. Context-aware development process

The development process presented in this work allows developers to exploit existing web services acting as reusable components towards the creation of composite web applications consisting of web service consumers. The web services comprising the application are adapted to context information separately without affecting the main application functionality. During the application execution this is guaranteed through a modularized architecture based on the interception of SOAP messages that encapsulate the WS requests and responses. The SOAP messages are modified accordingly to reflect the user and service context conditions. The final WS responses are included in the web application, whose implementation is based on the MVC architectural paradigm.

This separation of concerns is also maintained during the preceding stages of application analysis and design which leads to the mapping of the application design to platform specific code exploiting MDE tools (Fig. 1). The development methodology comprises of the following phases, which are detailed in the sections that follow:

- *Importation of existing service models in UML notation*: this step allows the reuse of existing services in the application under development.
- *Design of the application in the modeling environment*: the application is modeled in UML exploiting the imported models. The dependencies between context information and each business

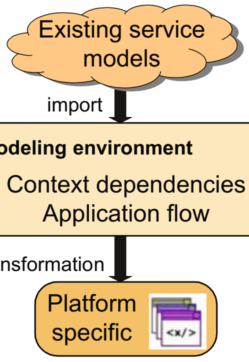


Fig. 1. Main Model-Driven Engineering steps.

[6] <http://www.eclipse.org/aspectj/>.

service present in the application, as well as the application flow indicating the order in which service operations are invoked, are also modeled here.
- *Mapping to platform specific code*: the result of the intermediate design phase is mapped to the source code and configuration files necessary for the application execution.

For the phases of the application design a number of UML profiles are introduced. These profiles – specific for the web service description, context information and presentation properties – are exploited during the modeling to capture different aspects of the application that need to be reflected in the resulting code (dependencies with context information, views and navigation, etc.). The imported service models of the first step also need to conform to the corresponding web service profile.

In the next section the application architecture is detailed with emphasis on the context adaptation of web services and their integration in the web application. Afterwards, the presentation of the introduced profiles and the development process in detail follows.

4. Context adaptation architecture

The supported applications of the context management architecture are designed as web applications following the MVC architectural pattern. Client requests are handled by the application controller, who performs appropriate calls to the business layer components, selects and populates the response views (Fig. 2). In the business layer, a number of web service clients implement locally the same interfaces as the ones found on the employed web service Web Services Description Language (WSDL) definitions. This approach allows for the direct reuse of existing business logic components exposed as web services, as well as the utilization of third party web services in the web application.

The adaptation to context is performed at the web service interface level. Specifically, the handler framework included in the JAX-WS specification [15] is exploited. Handlers are message processing modules that can be used to extend the functionality of a JAX-WS runtime system. Possible uses include message encryption or logging of web service activity. Handlers can reside on both client and server side, are organized in handler execution chains and may interact with incoming or outgoing messages based on their configuration. The extensions of the JAX-WS SOAP binding to the handler framework explicitly allow handlers to interact with the payload of SOAP envelopes. In this work, we employ handlers as means to adapt the content of exchanged messages based on context parameters.

The context management is performed through a modularized architecture (Fig. 3) that intercepts service requests and responses, retrieves the context information related to these messages and returns a modified message reflecting the context adaptation. The architecture is presented extensively in [11]. The message modification is carried out through a number of plugins. The context plugins communicate with the context sources that have direct access to the context information and alter the input (request) and output (response) messages to reflect the current user task and its environment. In this manner, the context adaptation is kept independent from the specific service implementation and clearly separated from the service logic. At the same time, the context adaptation logic can be altered at any time without making any changes to the web application, increasing thus the maintenance value of the proposed adaptation scheme. The context adaptation operations are performed by plugins, which are loaded automatically on runtime utilizing classloaders on the SOAP handler based on appropriate configuration files. Thus, the selection of the appropriate plugin, if needed, depends on the existing configuration, which can be altered at any time.

Specifically, the handler intercepts the SOAP message and may perform an adaptation, based on the message type (request or response) and the contents of the handler.xml configuration file, which associates the called web service operation with the appropriate plugin(s). The plugin(s) associated with the service is loaded by the handler. At this point the handler passes the SOAP Envelope to the plugin for processing. The plugin adapts the SOAP Envelope Body accordingly based on the contextual information returned from the context sources and returns it to the handler. The func-

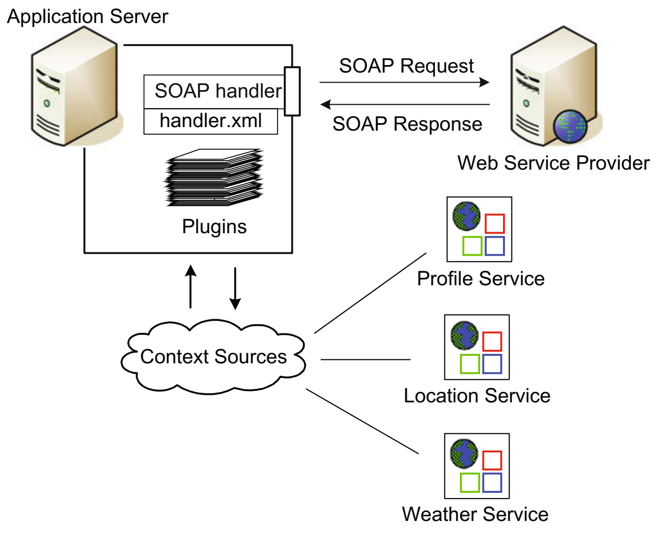


Fig. 3. The context adaptation mechanism.

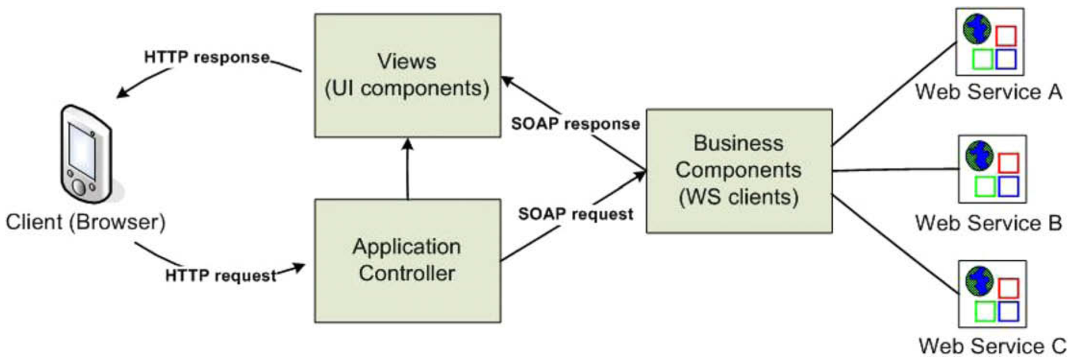


Fig. 2. Application design.

tionality of the handler remains the same in all cases of context management, i.e. message interception and loading of the appropriate plugin(s) which perform the modifications. All context adaptation logic lies in the development of the plugins and the configuration of the handler.xml file, both tasks clearly separated from the web application and the web services development and deployment procedure.

Each web service operation may be associated with one or more context plugins. The association indicates that one or more types of adaptation to context are to be performed depending on the number of plugins linked to the service. This mapping is represented in one configuration file: handler.xml. The main part of the corresponding XML Schema is depicted in Fig. 4. The file lists the available plugins (<plugin> elements) and their associations to business services (<association> elements). Each plugin identified by a unique ID (*id* attribute) is described by the name of the class implementing the plugin (*classname* attribute) and the package and jar file where it can be found (*pckg* and *jarfile* attributes, respectively). In each association element one specific plugin indicated by its id (<pluginEnd> element) is associated with an operation call (*operation* attribute in the <serviceEnd> element) of a specific business WS (*name* attribute). These associations show that the adaptation implemented in the plugin needs to be performed on the WS call. After the plugin(s) have been loaded, it is their role to retrieve the context information and apply it accordingly. The file along with the plugins implementing the adaptation logic can be updated during runtime without interrupting the application availability.

The available plugins are separated in the categories of *inPlugins* used to modify SOAP requests and *outPlugins* used to modify SOAP responses (*direction* attribute in the <pluginEnd> element). This is where the actual SOAP message modification takes place. The loaded plugins utilize one or more context sources and returns the modified message to the handler. The context sources are also web services that expose the context data of the underlying infrastructure (e.g. sensors, user profiles management system, etc.) to the plugins.

In all cases the adaptation is performed based on the current context information at the time of the service invocation without taking into account previous context values. With respect to the handling of historical context data, this would require an additional entity responsible for keeping record of the past context values. This entity can be modeled as an additional context source that takes input from existing context sources and produces a new context value. According to this convention, the resulting context value can be used to manipulate SOAP messages. Such a scheme can be supported by the presented approach with minor

```
<xs:element name="plugin" maxOccurs="unbounded">
  <xs:complexType>
     <xs:simpleContent>
       <xs:extension base="xs:string">
          <xs:attribute name="id" type="xs:string" use="required"/>
          <xs:attribute name="pckg" type="xs:string" use="required"/>
          <xs:attribute name="classname" type="xs:string"
use="required"/>
          <xs:attribute name="jarfile" type="xs:string"
use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
<xs:element name="association" maxOccurs="unbounded">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="serviceEnd">
      <xs:complexType>
        <xs:simpleContent>
         <xs:extension base="xs:string">
          <xs:attribute name="name" type="xs:string" use="required"/>
          <xs:attribute name="operation" type="xs:string"
use="required"/>
        </xs:extension>
       </xs:simpleContent>
      </xs:complexType>
    </xs:element>
    <xs:element name="pluginEnd">
     <xs:complexType>
      <xs:simpleContent>
       <xs:extension base="xs:string">
         <xs:attribute name="id" type="xs:string" use="required"/>
         <xs:attribute name="direction" type="xs:string"
use="required"/>
       </xs:extension>
      </xs:simpleContent>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
</xs:element>
```

Fig. 4. XML schema for the plugin association file.

modifications to the modeling process. However, this approach cannot be adopted, if the processing of past context data requires service behaviour modification that cannot be directly triggered by mapping past context to a new context value. In such a case it would be required from web services to keep context state, reason based on context state and alter their behaviour accordingly. This is an area for further study but it might be difficult to maintain the transparency between the web application and the adaptation mechanism at the modeling as well as at the implementation layer.

5. The modeling profiles

In order to be able to model a specific-purpose application in UML during the design process, an appropriate meta-model or profile needs to be defined first. Therefore, for the modeling of the complete web application and the context information, a number of profiles have been introduced. Meta-models extend the existing UML syntax to allow its use in domains of specific interest. Profiling is a specific meta-modeling technique that allows meta-classes from existing meta-models to be extended to adapt them for different purposes. Profiles are comprised of Stereotypes that extend existing meta-classes and Tag values that are standard meta-attributes. Having as starting point the defined profiles, compatible concrete application models can be constructed, whereas the profiles act as guidance for the code generation process. By using separate profiles we have the freedom of performing different combinations of service, context and presentation models. Their interaction is limited to a few relationships allowing to perform modifications on one model or to replace it without the need to re-model the whole application.

5.1. Web services profile

Web services interface descriptions in WSDL can be transferred in UML notation by applying stereotypes that correspond to WSDL elements. There exists a one-to-one mapping between the elements present in WSDL files and the web service model in terms of described port types, operations and messages. This means that if the service models in UML are not available they can be created or even generated using the original WSDL files. The intuitive web service profile widely adopted in similar ways (e.g. in Ref. [23]) is depicted on the left side of Fig. 5, where the values in brackets in each stereotype show the meta-classes it extends. The profile uses stereotypes that represent the web service port type (≪PortType≫), the supported service operations (≪WsOperation≫), the service messages (≪InMessage≫ and ≪OutMessage≫) that represent input and output messages respectively, the message parts (≪Part≫), which are the attributes found in web service messages, and finally any complex types (≪ValueObject≫) used as message parts.

5.2. Context meta-model

The web service profile can be combined with a context meta-model to form a complete profile that allows the modeling of the service and the context information, but also the dependencies between the two. In order to design the service and context model

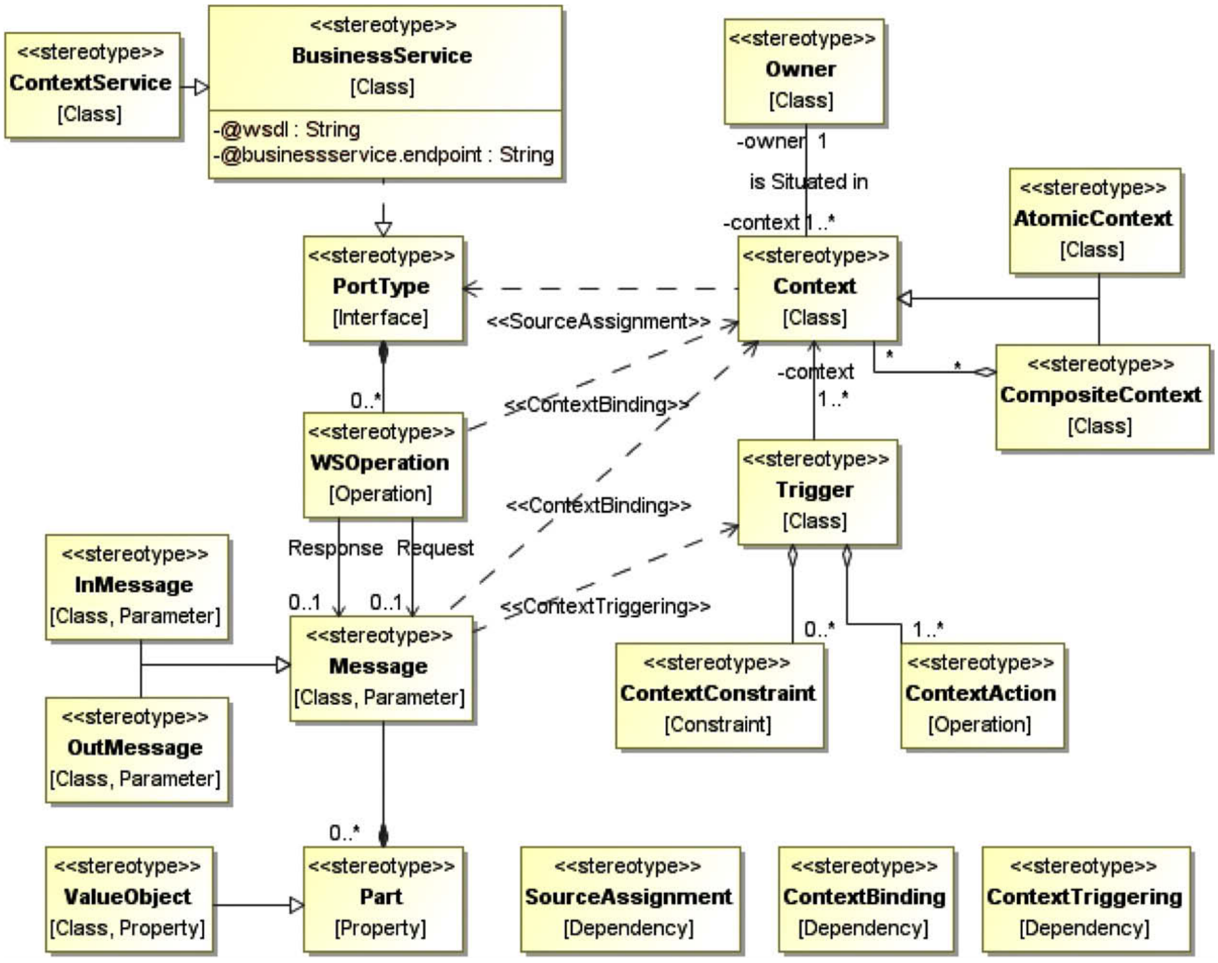
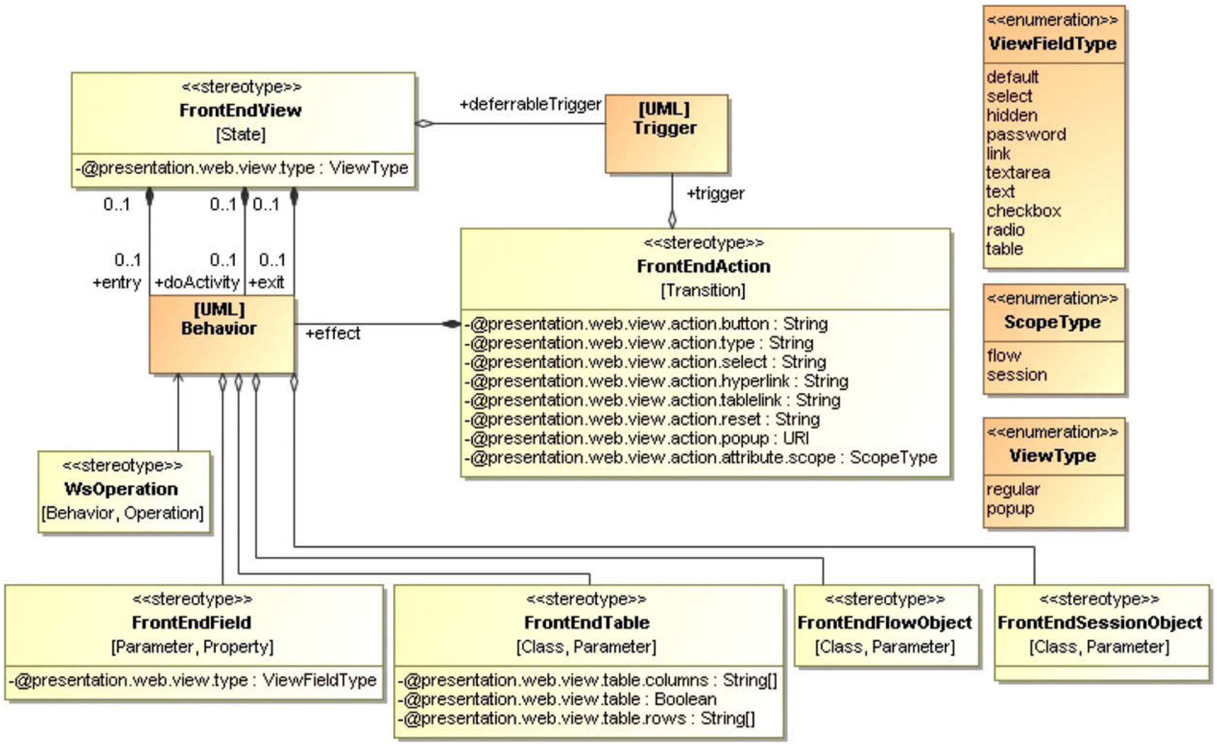

<<stereotype>> ContextService [Class]
<<stereotype>> BusinessService [Class]
-@wsdl : String
-@businessservice.endpoint : String
<<stereotype>> Owner [Class]
-owner 1
is Situated in
-context 1..*
<<stereotype>> AtomicContext [Class]
<<stereotype>> PortType [Interface]
<<stereotype>> Context [Class]
<<SourceAssignment>>
0..*
-context
1..*
*
*
<<stereotype>> CompositeContext [Class]
<<ContextBinding>>
<<stereotype>> WSOperation [Operation]
<<ContextBinding>>
<<stereotype>> Trigger [Class]
Response
Request
<<stereotype>> InMessage [Class, Parameter]
0..1
0..1
<<ContextTriggering>>
0..*
1..*
<<stereotype>> Message [Class, Parameter]
<<stereotype>> ContextConstraint [Constraint]
<<stereotype>> ContextAction [Operation]
<<stereotype>> OutMessage [Class, Parameter]
0..*
<<stereotype>> ValueObject [Class, Property]
<<stereotype>> Part [Property]
<<stereotype>> SourceAssignment [Dependency]
<<stereotype>> ContextBinding [Dependency]
<<stereotype>> ContextTriggering [Dependency]

Fig. 5. The web service and context meta-models.



<<stereotype>> FrontEndView [State]
-@presentation.web.view.type : ViewType
+deferrableTrigger
[UML] Trigger
+trigger
<<stereotype>> FrontEndAction [Transition]
-@presentation.web.view.action.button : String
-@presentation.web.view.action.type : String
-@presentation.web.view.action.select : String
-@presentation.web.view.action.hyperlink : String
-@presentation.web.view.action.tablelink : String
-@presentation.web.view.action.reset : String
-@presentation.web.view.action.popup : URI
-@presentation.web.view.action.attribute.scope : ScopeType
0..1 +entry
0..1 +doActivity
0..1 +exit
[UML] Behavior
+effect
<<stereotype>> WsOperation [Behavior, Operation]
<<enumeration>> ViewFieldType
default
select
hidden
password
link
textarea
text
checkbox
radio
table
<<enumeration>> ScopeType
flow
session
<<enumeration>> ViewType
regular
popup
<<stereotype>> FrontEndField [Parameter, Property]
-@presentation.web.view.type : ViewFieldType
<<stereotype>> FrontEndTable [Class, Parameter]
-@presentation.web.view.table.columns : String[]
-@presentation.web.view.table : Boolean
-@presentation.web.view.table.rows : String[]
<<stereotype>> FrontEndFlowObject [Class, Parameter]
<<stereotype>> FrontEndSessionObject [Class, Parameter]

Fig. 6. The presentation profile.

we will be using a meta-model based on a modified version of the ContextUML meta-model [20]. The meta-model is illustrated in Fig. 5. The relations of the web service and context meta-models are also shown.

The context meta-model extends the existing UML syntax, in order to enable the creation of context-aware service models. The ≪Context≫ stereotype is used for representing generic context information and is further distinguished into two subtypes: ≪AtomicContext≫ which is simple context information, directly provided by a context source and ≪CompositeContext≫ which is an aggregation of atomic or composite contexts. Composite context is formed using the values indicated by other context data, e.g. a high temperature value together with low intensity value for the wind can be aggregated to a good weather indication. The ≪SourceAssignment≫ dependency relationship associates context attributes with the respective context sources that provide values for these attributes. It is assumed that these context sources are also exposed as web services. The ≪Owner≫ stereotype associated with ≪Context≫ classes refers to the owner of the context information (e.g. User for user personal data and preferences, Device for device hardware and software characteristics, etc.).

The mechanisms that realize context-awareness are shown through the ≪ContextBinding≫ and ≪ContextTriggering≫ stereotyped relationships. ≪ContextBinding≫ models the direct association of context information with context-aware objects (operations and messages), which indicates the need for replacement of the service element with the corresponding context information. ≪ContextTriggering≫ models contextual adaptation depicting the dependency of service modification or execution on context information. A ≪ContextTriggering≫ element has two parts: a set of contextual constraints (≪ContextConstraint≫) expressed in UML's Object Constraint Language (OCL) [17] and an action (≪ContextAction≫) performed only if all constraints are met.

The context meta-model can be used for expressing three different cases of context adaptation supported by the proposed context management scheme detailed in the previous section:

- *Parameter injection*
  Parameter injection includes the population of operation parameters based on context. One or more parameters values of the request message are replaced with values related to atomic or composite context information (e.g. a parameter representing user language is set to the actual native language of the user instead of the reasonable default "English").
- *Operation selection*
  In this case the operation contained in the request message is changed to reflect the contextual needs. It is common for a service to aggregate a number of methods providing the same functionality in different ways or with different parameters. The service provider may support a mechanism that proactively selects one of the methods according to the context information found by the context sources. Assuming we have two different payment mechanisms each one provided as a single operation of a payment service the context adaptation can override the default selection based on the preference indicated in the a user profile or elsewhere.
- *Response manipulation*
  Response manipulation refers to the manipulation or the modification of service responses based on context. The adaptation is performed after the service has executed and constructed a response. This can include sorting or filtering operations (e.g. in the case of many results), converting (e.g. currency), etc.

The first two cases are expressed with ≪ContextBinding≫ relations, whereas ≪ContextTriggering≫ is used for the last one. Of course combinations of the above categories may also exist. For example, a parameter injection may be followed with a response manipulation.

It should be noted that the above adaptation cases are meaningful for web services, where it is essential to maintain both the SOAP request and response messages' integrity with respect to the signatures of the implemented WS operations. Indeed regarding the manipulation of a SOAP request, it is either

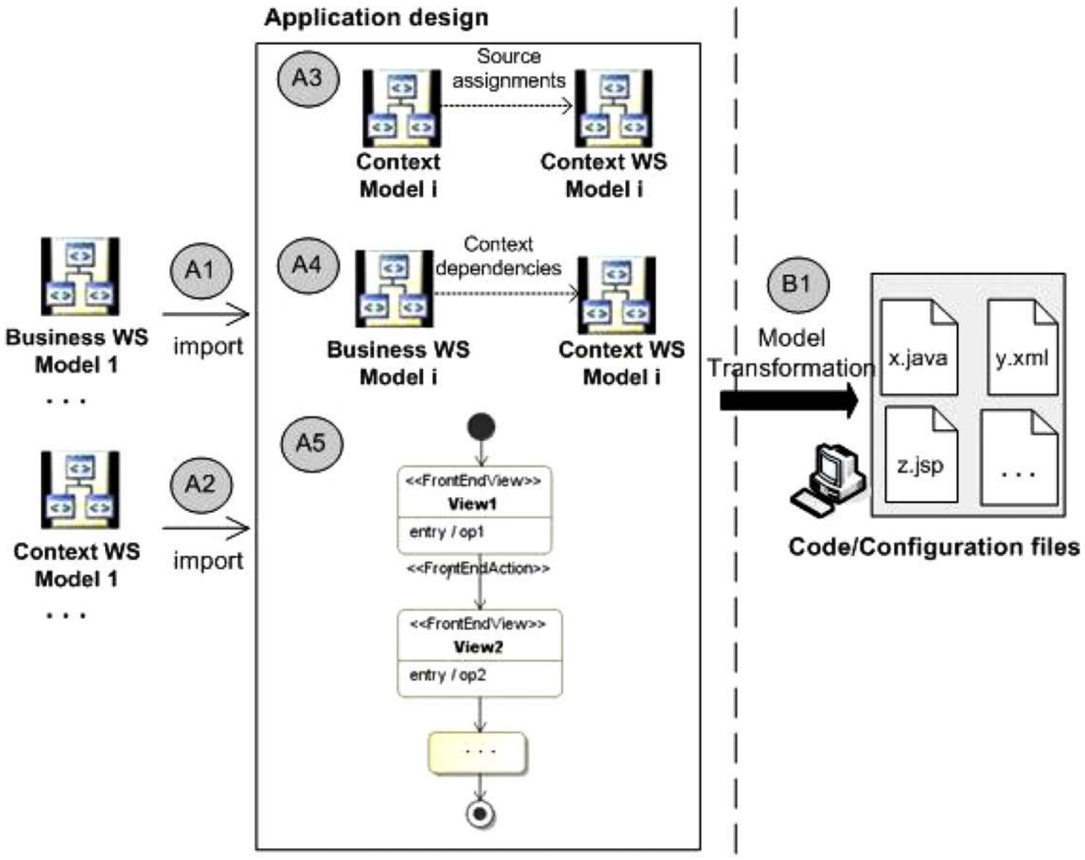


Fig. 7. The main steps of the context-aware service development methodology.

possible to change some (or all) of the parameter values with new, meaningful values depending on the available context data, or choose another implementation of the desired operation supported within the target web service. However, it would not be feasible to change the whole request structure which needs to adhere to the expected form as described in the WSDL description of the target service or change the initial target service of the message. Similar restrictions apply to SOAP responses, which also allow more complex manipulation procedures (e.g. sorting, filtering and converting) because of the nature of the response data.

5.3. Presentation profile

The presentation profile represents the presentation layer of the web application and is used for modeling the UI presentation, its flow and the application navigation properties. It is separate from the service and context profiles and can also be used for modeling web applications where no context adaptation mechanism exists. The profile, loosely based on a similar profile used by the AndroMDA code generator [1], is depicted in Fig. 6 represented together with a subset of the State Machine class diagram of the UML specification. The web application flow is modeled with UML state transition diagrams. State transition diagrams or state diagrams are used for modeling distinct states of application objects and dependencies between these states and are thus suitable for service flow modeling [8]. Moreover, state transition diagrams capture the application characteristics that can be found in MVC architectures and implementations like Apache Struts and Spring MVC. Especially the semantics used by the Spring Web Flow framework in order to describe the controller servlet workflow can directly be mapped to the notation elements of UML state transition diagrams.

Within the framework of an MVC web application, the aim of the presentation profile is to assist in the generation of the application controller as well as the view components. The profile comprises of a number of stereotypes and tags that have different functionality (e.g. some are used to show exactly how each response is mapped to elements of the web interface). These tags have the form: `@presentation.web.view.type = text/enumeration_type` and are visible in the diagram of the presentation profile along with some enumeration types that are used for the tag values.

The ≪FrontEndView≫ is the basic stereotype applicable on states. It refers to elements that are mapped to application view components. Front end views are associated with operations of the business web services of the application inserted as activities in the view (e.g. as state Entry, doActivity or Exit activities). The ≪FrontEndAction≫ stereotype can be applied to transitions between states and models an action performed on the front end (usually by the user) that triggers a transition to another state. Such actions can be a button press, a hyperlink, a field selection, etc. as specified in the `@presentation.web.view.action.xxx` tags. The ≪FrontEndField≫ stereotype represents a generic field in the view (of types usually found in web pages like text, radio button, check box, etc.), whereas ≪FrontEndTable≫ represents a table with specified row and column values. Apart from modeling states and transitions, there are cases where we want to maintain the values of some elements during the whole session, i.e. during the whole duration of the application(s) interaction with the user. The ≪FrontEndSessionObject≫ stereotype is used to mark these items, whereas ≪FrontEndFlowObject≫ has a shorter scope and is used to store values for elements that need to be passed between

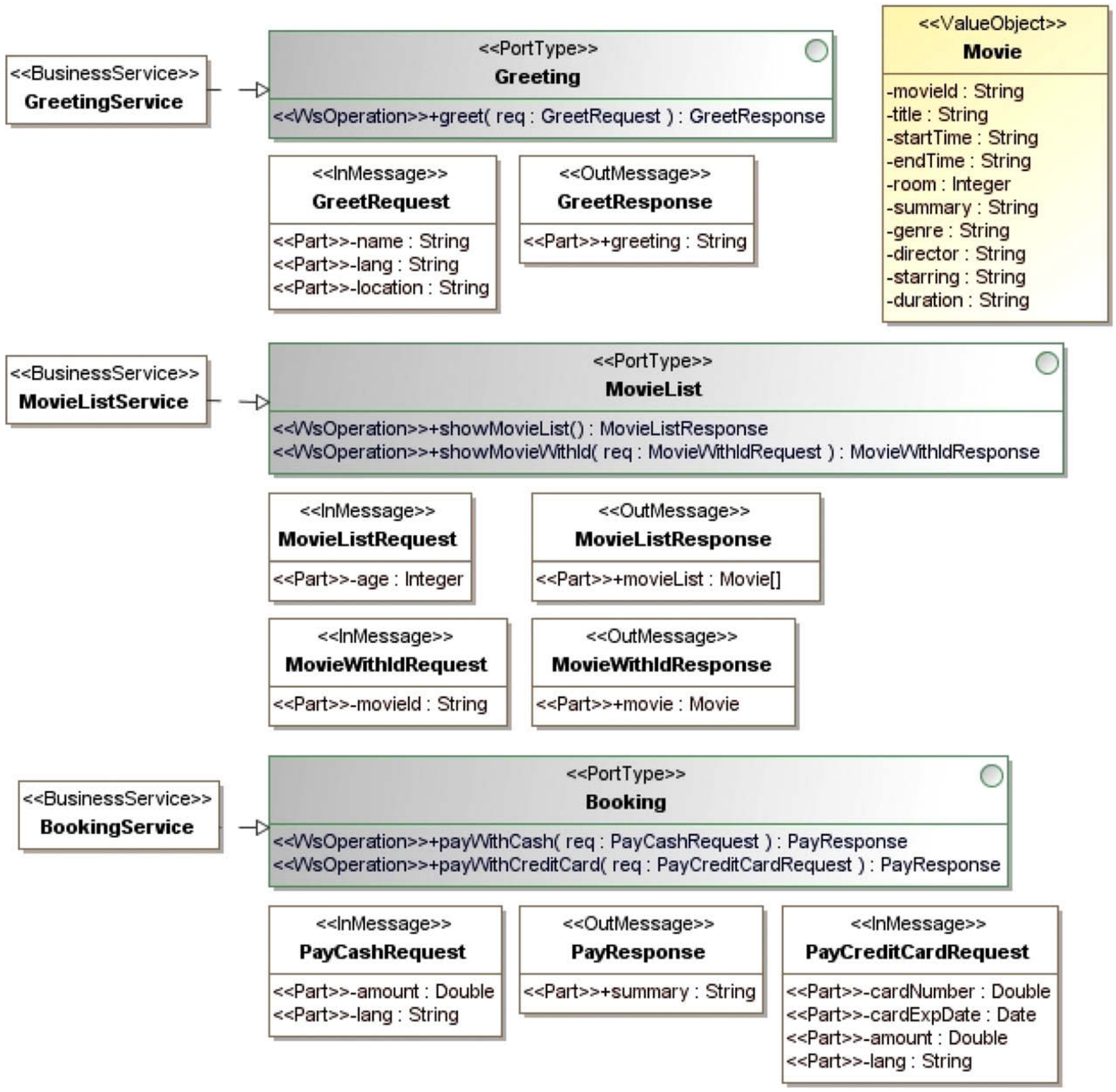


Fig. 8. Class diagram of the business service models for the cinema scenario.

a small number of views (usually representing a separate use case) and not the whole session.

6. The modeling process

In the modeling phase the developer constructs the concrete UML models that describe the desired application and adhere to the defined profiles. These models are then used in the code generation phase that follows.

The design process can be further broken down into the following actions in the specified order (left side of Fig. 7):

A1. Import of the business service models, if they exist. Otherwise, the WSDL descriptions of the web services are mapped to the model representation in UML following the notation of the web service profile.
A2. Import of the context service models, if they exist. Otherwise, the WSDL descriptions of the web services providing access to the context sources are mapped to the model representation in UML following the notation of the web service profile.
A3. Import (if it already exists) or design of the context model and the source assignment associations between the elements of the context model and the service models that provide access to the context sources designed in the previous step.
A4. Modeling of the relations between the two main models: the business logic and the context model. The type of context adaptation performed for each service (if any) needs to be defined first and then the dependencies are introduced in the form of ≪ContextBinding≫ and ≪ContextTriggering≫ relationships. Any ≪Trigger≫ classes and respective operations needed for response manipulation cases, which were not already available in the above service models, are also imported or designed.
A5. Design of the service flow in a UML state transition diagram using the notation of the presentation profile. The models of the business services are composed to form a web application comprised of states that correspond to application views and transitions that may be associated with parts of web service operation messages. Each state in the state tran-

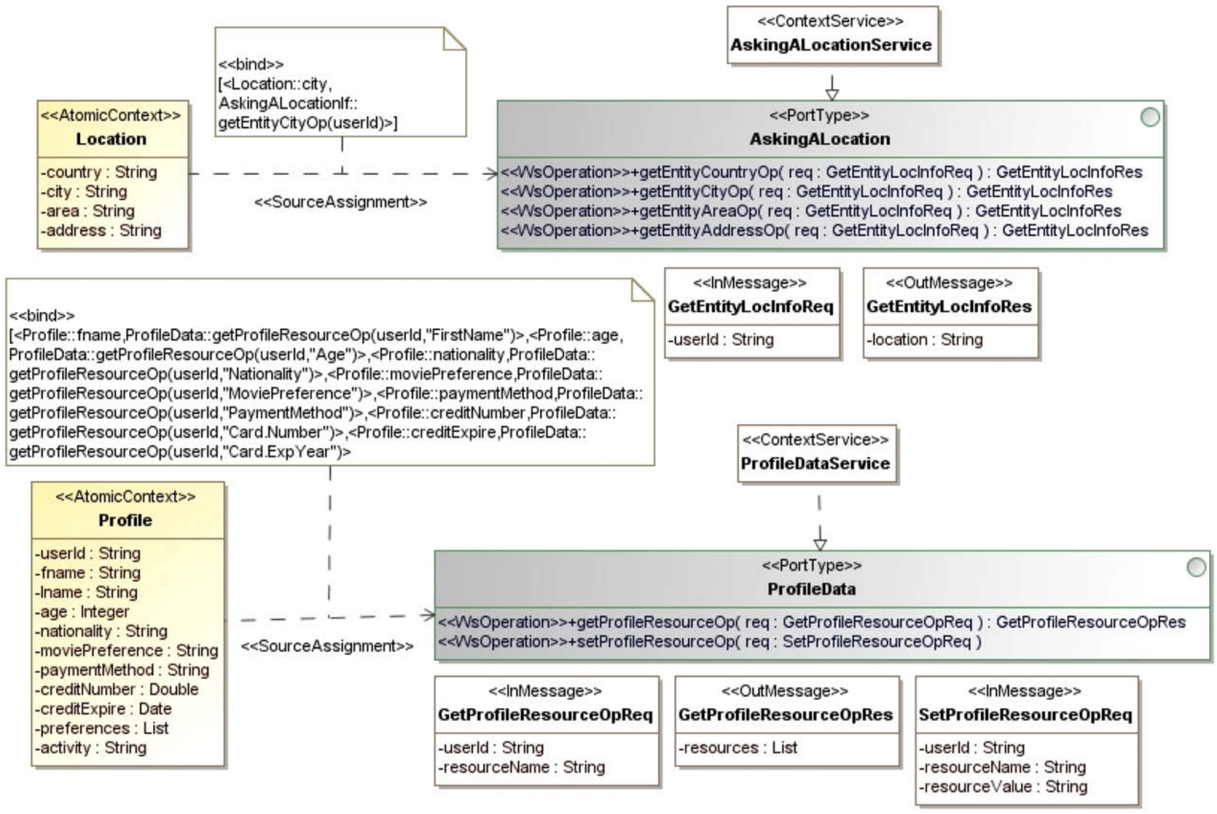

<<bind>>
[<Location::city, AskingALocationIf:: getEntityCityOp(userId)>]
<<ContextService>>
AskingALocationService
<<AtomicContext>>
Location
-country : String
-city : String
-area : String
-address : String
<<SourceAssignment>>
<<PortType>>
AskingALocation
<<WsOperation>>+getEntityCountryOp(req : GetEntityLocInfoReq) : GetEntityLocInfoRes
<<WsOperation>>+getEntityCityOp(req : GetEntityLocInfoReq) : GetEntityLocInfoRes
<<WsOperation>>+getEntityAreaOp(req : GetEntityLocInfoReq) : GetEntityLocInfoRes
<<WsOperation>>+getEntityAddressOp(req : GetEntityLocInfoReq) : GetEntityLocInfoRes
<<InMessage>>
GetEntityLocInfoReq
-userId : String
<<OutMessage>>
GetEntityLocInfoRes
-location : String
<<bind>>
[<Profile::fname,ProfileData::getProfileResourceOp(userId,"FirstName")>,<Profile::age, ProfileData::getProfileResourceOp(userId,"Age")>,<Profile::nationality,ProfileData:: getProfileResourceOp(userId,"Nationality")>,<Profile::moviePreference,ProfileData:: getProfileResourceOp(userId,"MoviePreference")>,<Profile::paymentMethod,ProfileData:: getProfileResourceOp(userId,"PaymentMethod")>,<Profile::creditNumber,ProfileData:: getProfileResourceOp(userId,"Card.Number")>,<Profile::creditExpire,ProfileData:: getProfileResourceOp(userId,"Card.ExpYear")>
<<ContextService>>
ProfileDataService
<<AtomicContext>>
Profile
-userId : String
-fname : String
-lname : String
-age : Integer
-nationality : String
-moviePreference : String
-paymentMethod : String
-creditNumber : Double
-creditExpire : Date
-preferences : List
-activity : String
<<SourceAssignment>>
<<PortType>>
ProfileData
<<WsOperation>>+getProfileResourceOp(req : GetProfileResourceOpReq) : GetProfileResourceOpRes
<<WsOperation>>+setProfileResourceOp(req : SetProfileResourceOpReq)
<<InMessage>>
GetProfileResourceOpReq
-userId : String
-resourceName : String
<<OutMessage>>
GetProfileResourceOpRes
-resources : List
<<InMessage>>
SetProfileResourceOpReq
-userId : String
-resourceName : String
-resourceValue : String

Fig. 9. Class diagram of the context service models and relations with the context model for the cinema scenario.

sition diagram corresponds to one application view which is associated with one or more WS operation calls expressed as state activities (call operation action). The WS operations belong to the previously imported business service models. The defined activity is also associated with parameters that depict which operation responses should be rendered in the application views and how (as indicated by the ≪FrontEndViewField≫ and ≪FrontEndTable≫ tags). Similarly, parameters are associated with the transitions between states as *Effect* activities. These parameters indicate which attribute values should be maintained for the whole flow or session duration (≪FrontEndFlowObject≫ and ≪FrontEndSessionObject≫ stereotyped parameters), whereas the tags on the transition show exactly which view elements trigger the transition action (e.g. buttons or hyperlinks).

The above steps can be performed exploiting any modeling tool that supports UML diagrams and introduction of UML profiles. The modeling tool also needs to allow the export of the model in its XMI or Eclipse Modeling Framework (EMF) UML2 [3,7] format. Examples of such tools are MagicDraw[7] and ArgoUML.[8] For the business service and context service models third party services functioning as service components can also be exploited.

7. The transformation process

Once the application has been sufficiently modeled the code generation procedure follows. The transformation is based on the stereotypes and tag values of the UML web service, context and presentation profiles used in the design of the web application. This way the model is mapped to source code and any necessary implementation specific accompanying files to provide a fully functional application (right side of Fig. 7). A transformation tool consisting of a model parser and a code generator has been implemented. As a demonstration case, the implemented code generator targets our context adaptation architecture and Spring MVC and Web Flow web application frameworks. However, the mapping can also be performed considering target implementations based on different web development technologies (e.g. Apache Struts), since the fundamental web application development principles remain the same in all cases and the profiles are generic and abstract enough to support different kind of code mappings.

EMF and the UML2 library that implements the UML 2.0 metamodel in Java, also offered by the Eclipse platform, have been used for the model parsing part together with a number of dedicated tools developed during our work. More precisely, for the model parsing all diagrams are traversed in their EMF format that can be exported either from the modeling tool or from the XMI representation of the model. Afterwards, the presence of specific stereotypes on the model elements read during the parsing (≪BusinessService≫, ≪FrontEndView≫, etc.) triggers the execution of Apache Velocity[9] templates that are responsible for the generation of the actual source code and other files. Velocity offers a variety of tools, but works at its core as a template engine and is widely used for code generation to various file formats (source code, HTML or XML files, etc.). Velocity is implemented in Java and thus a number of Java objects can be added in Velocity Context and then be referred to from Velocity templates. Velocity templates have been designed for all implementation specific files that are to

[7] <http://www.magicdraw.com/>.
[8] <http://argouml.tigris.org/>.
[9] <http://velocity.apache.org/>.

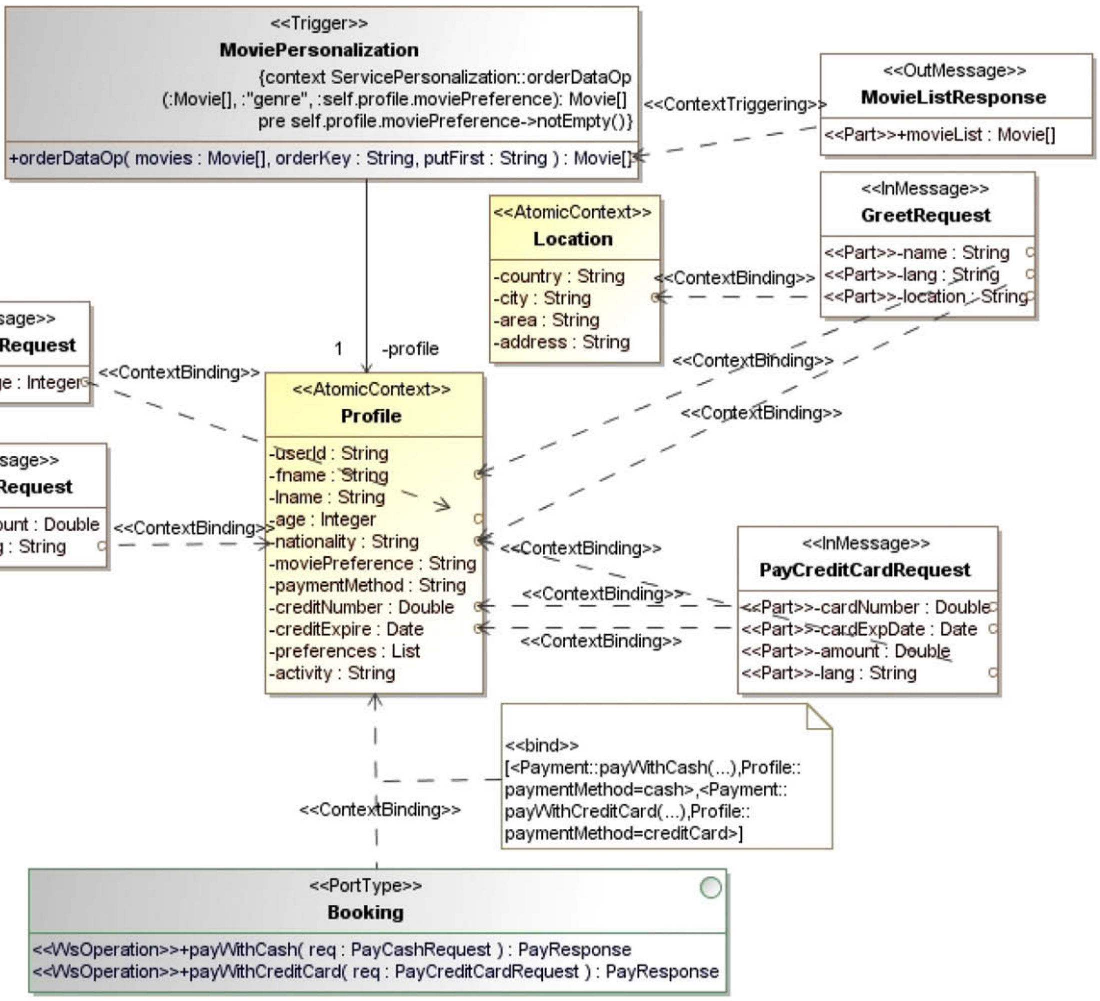


Fig. 10. Dependencies between the business and the context model for the cinema scenario.

be generated (JSP files, web service clients in Java, plugin code in Java, etc.). By including in these templates corresponding operations that access the model information, the generator can add to the source code and configuration files the information present in the model classes, interfaces, attributes, operations, associations, stereotypes and tag values (e.g. the `@presentation.web.view.table.columns` tag value is used to inject to the application view the names and information to be used in the table rendering).

Specifically, the mapping of model to code based on the applied stereotype definitions and tags is performed as follows:

- *Bean generation*
  Java classes with setters and getters methods for their attributes are generated for each operation message (≪InMessage≫ and ≪OutMessage≫ stereotyped classes) found in the model parsing. For the complex types included in the web service operation parameters or return types respective beans are also generated. This information can be found in the WSDL file, and hence in the class diagrams used for the application design.
- *Web service clients*
  A separate client is generated for each of the imported ≪BusinessService≫ stereotyped classes included in the model. These are the clients needed to call the WS operations, so the clients are generated in order to allow calls to the appropriate operations through the behaviours defined in the states of the state transition diagram. This way, web service operations are exposed to the client as local methods.
- *Plugins code*
  Appropriate plugins are generated for each ≪BusinessService≫ class based on the indirect dependencies specified between the web service elements (operations, in and out messages, etc.) and the context sources. These plugins are generated as children of `AbstractQueryPlugin`, which is the basic plugin class already provided by the handler of the context management framework. In order to generate the plugins all port types related to business services are parsed. The dependencies of the elements of the services implementing the port types are found and the appropriate plugins that correspond to each context adaptation case are generated:
  1. <porttypename><operationname>InPlugin.java files are generated for each parameter injection case when the corresponding ≪ContextBinding≫ dependency is parsed in the model. For each parameter of the request message the replacement with the actual value retrieved through the context sources is made (if such a dependency exists).
  2. <porttypename>InPlugin.java files are generated for each operation selection case that is found through the dependency that exists between the port type and the context item in the model. The called operation as specified in the SOAP request is changed to the operation that needs to be called according to the context dependency.
  3. <porttypename><operationname>OutPlugin.java files are generated for each response manipulation case when the ≪ContextTriggering≫ relationship is found in the model.

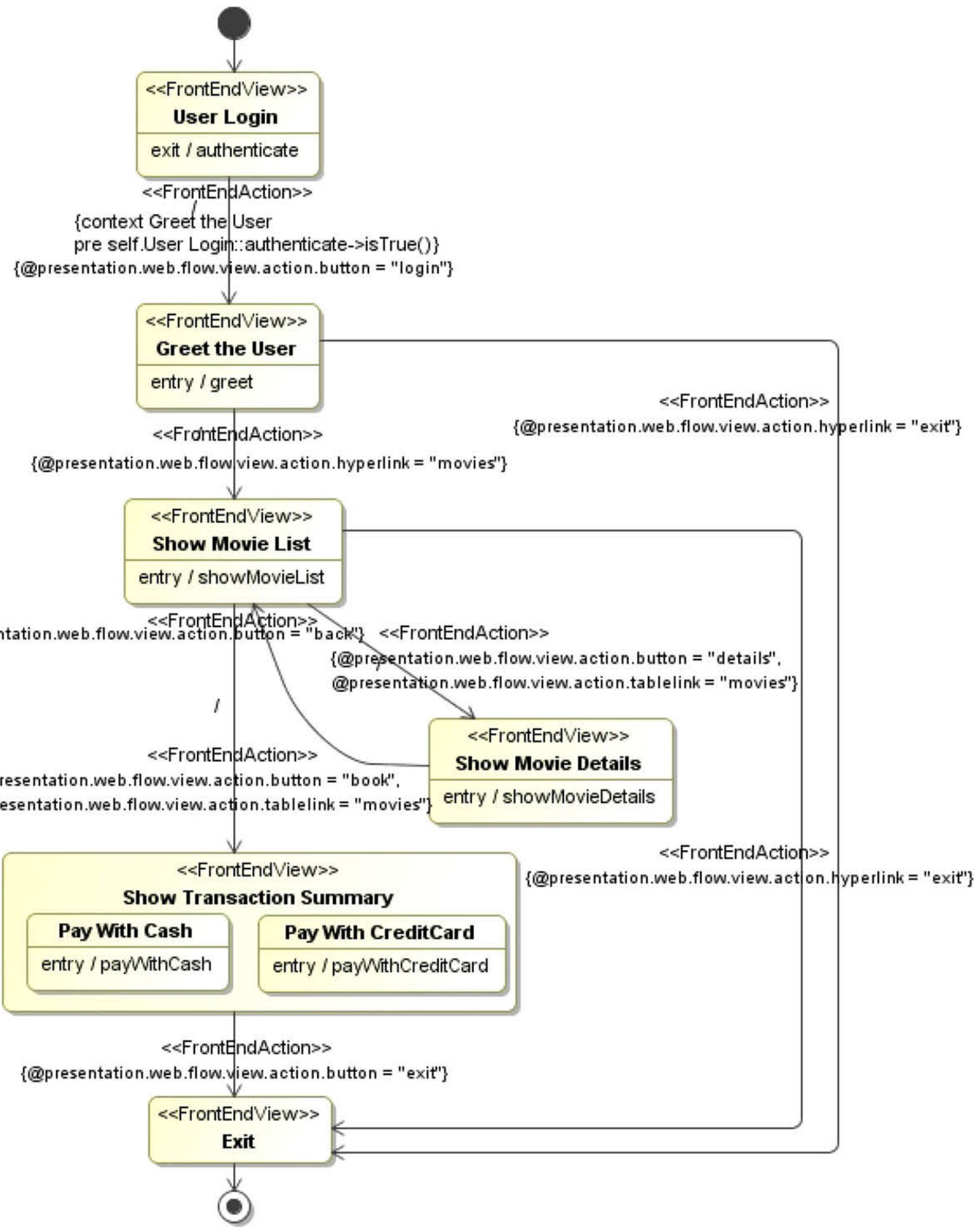
The SOAP response elements are modified through the triggered operation (they are usually reordered or filtered according to specified constraints).

- *Configuration files*

A flow.xml file representing the states and transitions and a flow-beans.xml file describing the application context (as defined in Spring MVC and Spring Web Flow) are generated for the state transition diagram. For each plugin generated appropriate entries are put in the handler.xml file containing the plugin and the association definitions. The other configuration files (e.g. web.xml, application-webflow-config.xml and application-servlet-config.xml) are also generated using corresponding templates or filled with the default content used in Spring WebFlow.

- *View files*

One JSP file is generated for every ≪FrontEndView≫ stereotyped state. The JSP generation includes the references to the obtained objects that represent the business layer data forwarded by the controller to the view, as well as the events



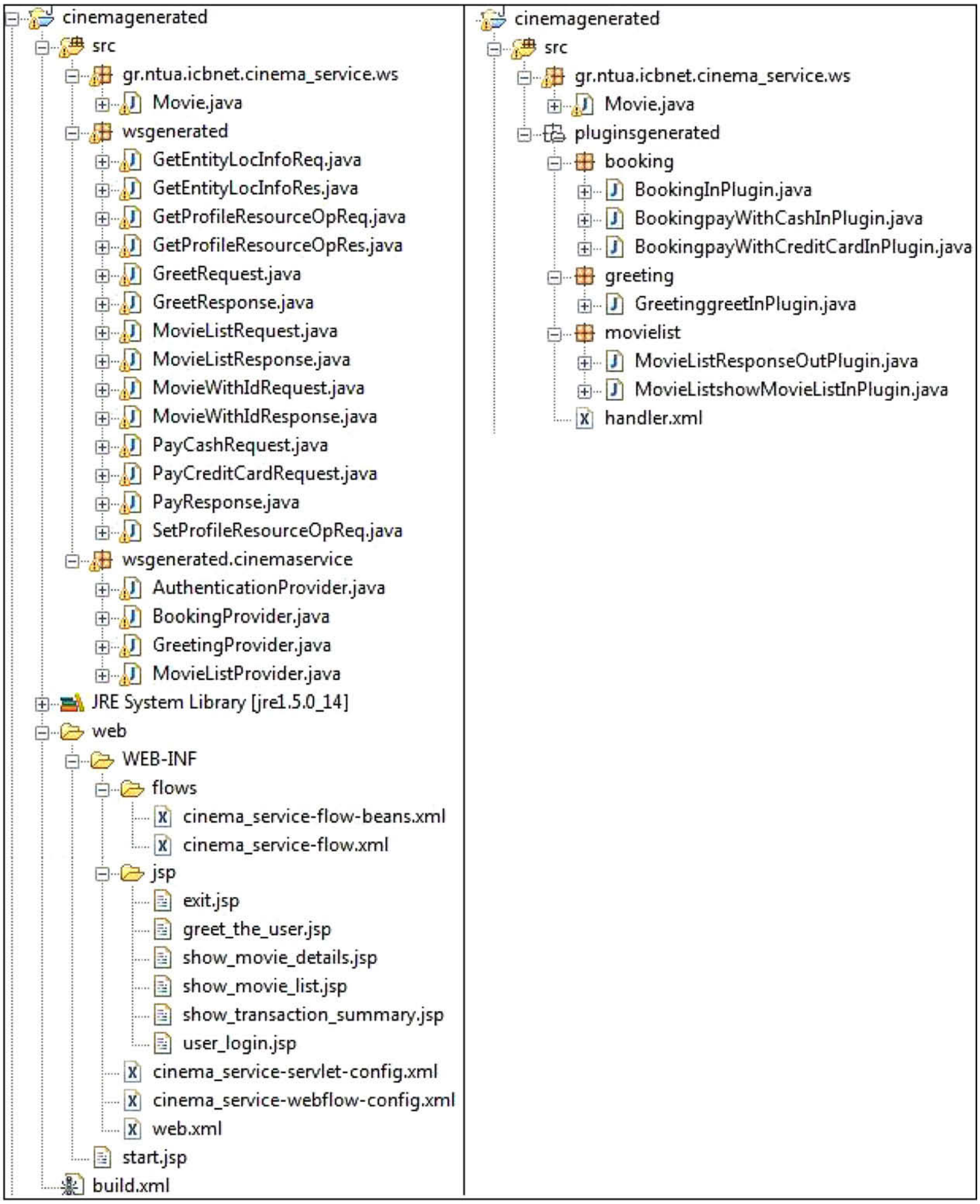
<<FrontEndView>>
User Login
exit / authenticate
<<FrontEndAction>>
/
{context Greet the User
pre self.User Login::authenticate->isTrue()}
{@presentation.web.flow.view.action.button = "login"}
<<FrontEndView>>
Greet the User
entry / greet
<<FrontEndAction>>
{@presentation.web.flow.view.action.hyperlink = "exit"}
<<FrontEndAction>>
{@presentation.web.flow.view.action.hyperlink = "movies"}
<<FrontEndView>>
Show Movie List
entry / showMovieList
<<FrontEndAction>>
{@presentation.web.flow.view.action.button = "back"}
<<FrontEndAction>>
{@presentation.web.flow.view.action.button = "details",
@presentation.web.flow.view.action.tablelink = "movies"}
<<FrontEndView>>
Show Movie Details
entry / showMovieDetails
<<FrontEndAction>>
{@presentation.web.flow.view.action.button = "book",
@presentation.web.flow.view.action.tablelink = "movies"}
<<FrontEndAction>>
{@presentation.web.flow.view.action.hyperlink = "exit"}
<<FrontEndView>>
Show Transaction Summary
Pay With Cash
entry / payWithCash
Pay With CreditCard
entry / payWithCreditCard
<<FrontEndAction>>
{@presentation.web.flow.view.action.button = "exit"}
<<FrontEndView>>
Exit

Fig. 11. The flow diagram of the cinema application.

leading to the defined transitions, either as form buttons or as hyperlinks. This information is mostly gained through the state diagram tag values on state and transition elements. Note that it is not the objective of this work to specify means for the design of the views; our goal lies in the automatic generation of minimal views which can be enriched independently by a web designer.

8. Demonstration scenario

In order to validate the proposed architecture and development methodology a number of test cases have been carried out during the generator development and model definition in an iterative process. The initial testing involved the introduction of abstract web services that did not provide concrete functionality (e.g. web



```
cinemagenerated
  src
    gr.ntua.icbnet.cinema_service.ws
      Movie.java
    wsgenerated
      GetEntityLocInfoReq.java
      GetEntityLocInfoRes.java
      GetProfileResourceOpReq.java
      GetProfileResourceOpRes.java
      GreetRequest.java
      GreetResponse.java
      MovieListRequest.java
      MovieListResponse.java
      MovieWithIdRequest.java
      MovieWithIdResponse.java
      PayCashRequest.java
      PayCreditCardRequest.java
      PayResponse.java
      SetProfileResourceOpReq.java
    wsgenerated.cinemaservice
      AuthenticationProvider.java
      BookingProvider.java
      GreetingProvider.java
      MovieListProvider.java
  JRE System Library [jre1.5.0_14]
  web
    WEB-INF
      flows
        cinema_service-flow-beans.xml
        cinema_service-flow.xml
      jsp
        exit.jsp
        greet_the_user.jsp
        show_movie_details.jsp
        show_movie_list.jsp
        show_transaction_summary.jsp
        user_login.jsp
      cinema_service-servlet-config.xml
      cinema_service-webflow-config.xml
      web.xml
    start.jsp
  build.xml
```

```
cinemagenerated
  src
    gr.ntua.icbnet.cinema_service.ws
      Movie.java
    pluginsgenerated
      booking
        BookingInPlugin.java
        BookingpayWithCashInPlugin.java
        BookingpayWithCreditCardInPlugin.java
      greeting
        GreetinggreetInPlugin.java
      movielist
        MovieListResponseOutPlugin.java
        MovieListshowMovieListInPlugin.java
      handler.xml
```

Fig. 12. Code generated for the cinema application and the context adaptation plugins.

service A with operations A to E). In the next phase the development of applications consisting of one business service was tested and finally the combination of more web services was included, where the main validation performed lied in the proper behaviour of the generated application. In this section the demonstration of a specific development scenario is used.

8.1. Scenario description

The following scenario is considered: a developer has been assigned with the task of the creation of an application offered to all visitors of the "AllCinemas" cinema area through their handheld devices. The goal is to offer a context-aware web application in order to provide the cinema customers with an overview of the scheduled movies and means to perform a reservation. The developer may exploit existing third party web services or web services exposing functionality of the cinema intranet application. The application integrates three web services; a Greeting service that is used to display a welcome message, a MovieList service that displays to the user the available movies and details for each one of them and a Payment service that allows the user to pay for his tickets. Each of these services is adapted to context. The Greeting service is injected with contextual parameters that represent the user's name, his native language and his current location. The first two are retrieved from the user's public profile, whereas the last one is taken from a location service. The MovieList service operation that displays the available movies is adapted first by taking into account the requester's age in the list of movies to be returned and then by manipulating the service response in order to sort the results based on the film genre preference specified in the user's profile. Finally, the Payment service adaptation is a case of selection of the appropriate operation (payWithCash or payWithCreditCard) based on the user preferences, whereas a parameter indicating the user language is injected in the respective operation requests as in the GreetingService case.

In the next sections it is shown how this application can be developed by exploiting the proposed architecture and the model-driven development methodology.

8.2. Cinema scenario modeling

For the modeling phase of the use case example the Magicdraw modeling environment was exploited. The business service models each corresponding to a different web service are shown in Fig. 8. The Movie complex type represents information on movie objects and can be extracted from the WSDL description.

The models of the context services follow the same notation, but refer to providers of context information and not business functionality. These services are depicted in Fig. 9 along with the corresponding source assignment relations with the context model that consists of the following context information:

- *Location*: expresses a specific location in terms of country, city, city area and address
- *Profile*: expresses a user profile containing information on the user and his preferences (e.g. nationality, current activity, etc.)

```
<?xml version="1.0" encoding="UTF-8"?>
<plugins>
       <plugin id="parinj1" pckg="pluginsgenerated.greeting"
classname="GreetingGreetInPlugin" jarfile="GreetingPl.jar"/>
       <plugin id="resman2" pckg="pluginsgenerated.movielist"
classname="MovieListResponseOutPlugin" jarfile="MovieListPl.jar"/>
       <plugin id="parinj3" pckg="pluginsgenerated.movielist"
classname="MovieListshowMovieListInPlugin" jarfile="MovieListPl.jar"/>
       <plugin id="opsel4" pckg="pluginsgenerated.booking"
classname="BookingInPlugin" jarfile="BookingPl.jar"/>
       <plugin id="parinj5" pckg=" pluginsgenerated.booking"
classname="BookingpayWithCashInPlugin" jarfile="BookingPl.jar"/>
       <plugin id="parinj6" pckg=" pluginsgenerated.booking"
classname="BookingpayWithCreditCardInPlugin" jarfile="BookingPl.jar"/>
       <association>
          <serviceEnd name="GreetingService" operation="greet"/>
          <pluginEnd id="parinj1" direction="in"/>
       </association>
       <association>
          <serviceEnd name="MovieListService" operation="showMovieList"/>
          <pluginEnd id="resman2" direction="in"/>
       </association>
       <association>
          <serviceEnd name="MovieListService" operation="showMovieList"/>
          <pluginEnd id="resman2" direction="out"/>
       </association>
       <association>
          <serviceEnd name="MovieListService" operation="showMovieList"/>
          <pluginEnd id="parinj3" direction="in"/>
       </association>
       <association>
          <serviceEnd name="BookingService" operation="payWithCash"/>
          <pluginEnd id="opsel4" direction="in"/>
       </association>
       <association>
          <serviceEnd name="BookingService" operation="payWithCash"/>
          <pluginEnd id="parinj5" direction="in"/>
       </association>
       <association>
          <serviceEnd name="BookingService" operation="payWithCreditCard"/>
          <pluginEnd id="parinj6" direction="in"/>
       </association>
</plugins>
```

Fig. 13. The handler.xml file for the cinema scenario.

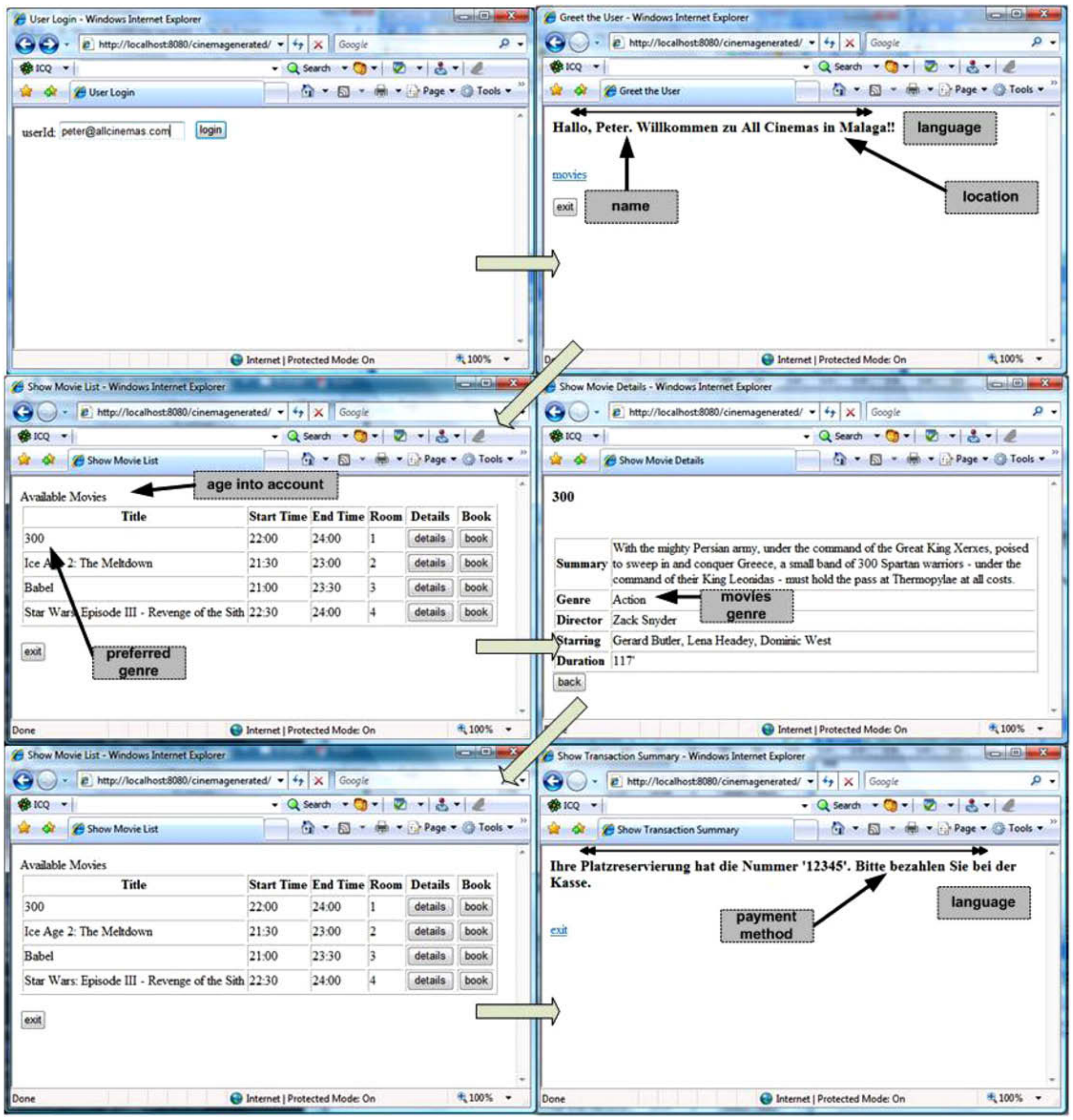
In the case of multiple ≪SourceAssignment≫ dependency relationships, instead of relating each part of the context information with the corresponding operation, the ≪bind≫ notation introduced in [25] is used for simplification purposes, while this information is inserted as comment on the dependency, in order to be visible for the code generation process. This notation is also used when modeling the dependencies between the business logic and the context model in the next step. The `@businessservice.endpoint` tags of the ≪BusinessService≫ classes – that present the URL where the web service is available or another access URL in case different kind of services are used – have been omitted from the above class diagrams for presentation purposes. An example of such a tag value is:

`@businessservice.endpoint = http://contextmda.icbnet.ntua.gr:8080/axis2/services/MovieListService.`

The result of the modeling step regarding the dependencies between the two main models can be found in Fig. 10. Cases of all three context adaptation categories as well as combinations of them are visible:

- Parameter injection can be seen in the ≪ContextBinding≫ dependencies deriving from the GreetRequest, MovieListRequest, PayCashRequest and PayCreditCardRequest request messages.
- Operation selection is found in the dependency between the Booking service and the Profile. The selected operation must correspond to the value of the paymentMethod attribute stored in the user profile as expressed in the ≪bind≫ comment.
- Response manipulation is used in the ≪ContextTriggering≫ dependency coming out from the MovieListResponse message. The dependency targets the MoviePersonalization trigger and more precisely its operation orderDataOp. The operation is called only if the moviePreference attribute in the user profile has a non-empty value as expressed by the precondition in the constraint which will also be parsed during the generation to assist in determining the behaviour of the generated plugin. Through this action the genre corresponding to the user favourite movies are placed on top of the movie list presented.

The results of the preceding modeling steps are used to perform the last step of the application flow modeling (Fig. 11). The first application view is the user login page (User Login state). When the login button is pressed, an operation for the user authentication to the system is called. In case the user is authenticated successfully (constraint on the transition) the transition to the greet view is performed (Greet the User state). The greet operation of the GreetingService is called before entering the view and the


User Login - Windows Internet Explorer
http://localhost:8080/cinemagenerated/
User Login
userId: peter@allcinemas.com
login
Greet the User - Windows Internet Explorer
Greet the User
language
name
location
movies
exit
Show Movie List - Windows Internet Explorer
Show Movie List
age into account
preferred genre
Show Movie Details - Windows Internet Explorer
Show Movie Details
movies genre
back
Show Transaction Summary - Windows Internet Explorer
Show Transaction Summary
payment method
Internet | Protected Mode: On
Done

Hallo, Peter. Willkommen zu All Cinemas in Malaga!!

Available Movies

Title	Start Time	End Time	Room	Details	Book
300	22:00	24:00	1	details	book
Ice Age 2: The Meltdown	21:30	23:00	2	details	book
Babel	21:00	23:30	3	details	book
Star Wars: Episode III - Revenge of the Sith	22:30	24:00	4	details	book

300

Summary	With the mighty Persian army, under the command of the Great King Xerxes, poised to sweep in and conquer Greece, a small band of 300 Spartan warriors - under the command of their King Leonidas - must hold the pass at Thermopylae at all costs.
Genre	Action
Director	Zack Snyder
Starring	Gerard Butler, Lena Headey, Dominic West
Duration	117'

Available Movies

Title	Start Time	End Time	Room	Details	Book
300	22:00	24:00	1	details	book
Ice Age 2: The Meltdown	21:30	23:00	2	details	book
Babel	21:00	23:30	3	details	book
Star Wars: Episode III - Revenge of the Sith	22:30	24:00	4	details	book

Ihre Platzreservierung hat die Nummer '12345'. Bitte bezahlen Sie bei der Kasse.

Fig. 14. Demonstration of the running application.

response message is displayed on user's screen. At this point two transitions are defined: by pressing the movies link the showMovieList operation is called and the transition to the next view is made (Show Movie List state), whereas transition to Exit state is triggered by pressing the exit link. While on the movie list view, a transition can be made either towards a view showing movie details (Show Movie Details state) or towards the composite booking view (Show Transaction Summary state). The booking view state contains the two states with both operations of the BookingService whose selection depends on the operation selection context adaptation. Apparently, many additional tag values are needed to model the service flow in full – especially for the objects flowing on transitions. Most of these are not visible in Fig. 11, because the modeling software does not display all UML artifacts' properties on the corresponding diagrams.

8.3. Cinema application mapping

Having as input the above diagrams for the use case scenario web application the corresponding code is generated. Fig. 12 illustrates the generated files for the cinema example. These include the web service clients and the beans for complex types included in the web service operations, the web application deployment descriptor and Spring MVC and Web Flow related configuration files, the handler.xml and the context adaptation plugins. The handler source code is application independent and does not need to be generated.

The handler.xml configuration file that includes the plugins and the associations is shown in Fig. 13. The references to the jar files refer to the result of the execution of the corresponding build.xml ant script (also generated to assist the application and plugin deployment).

A demonstration of the running application for the cinema scenario for a specific context adaptation case is depicted in Fig. 14. After the user logs in the greeting message response of the GreetingService appears in the user's language (German) taking into account the user's name (Peter) and location (Malaga). When the user proceeds to the next page the movie list is retrieved from the movie list service taking into account the requester's age (above 18) and presented to the user ordered according to the user movie genre preferences (action). From the main movie list page the user can view more details for the movie of his choice or book a ticket using the payment method preferred by the user (cash). The message returned by the payment method is also presented in the user language (German).

9. Conclusions

In this article, an architecture for the context adaptation of web applications consisting of web services and a model-driven methodology for the development of such context-aware composite applications have been presented. Starting from the application modeling in UML, a composite web application targeting different implementation platforms is generated. The application design on the modeling level is kept – at a great extent – independent from specific platform implementations and flexible enough to allow the introduction of different code specific mappings. During both the application development and execution phases the service logic and the context adaptation are kept independent providing flexibility and facilitating the application maintenance. For demonstration purposes, the application generation in this paper is targeting widely used platforms and frameworks suitable for the proposed architecture. The described approach combines and extends some of the best qualities of the related work in a common framework:

- Context adaptation is performed on a service interface level and is kept moreover client independent, requiring no specific software on the client-side.
- The modeling is performed in UML notation instead of any proprietary DSL, maintaining compatibility with off-the-shelf UML modeling tools.
- Context adaptation and application modeling are tackled simultaneously resulting in the generation of context-aware web applications without, however, introducing undesirable binding of the application and context adaptation functionality.

The prototype implementation of the presented work, along with the demonstrated example, is available online.[10] We also aim at performing further validation of the approach (e.g. using functionalities offered by online web services[11]) and evaluate it by introducing a number of development metrics (e.g. generation time for different levels of model complexity, number and extent of source code sections where developer intervention may be required, etc.).

In future work we will explore the use of historical context information in the context adaptation of the web application and how it can be handled in a generic context-aware framework. The framework will also address the issue of context storage allowing the storage and retrieval of past context information, as well as the intelligent combination of these past contexts into a resulting composite context. This way other entities can exploit the added information to adapt their behaviour to previous context values or a series of events indicating the user situation context (e.g. series of locations visited may indicate that the user is on a trip). The area of past context values requires however further study following the concerns raised in the paper.

Acknowledgement

G. Kapitsaki is supported by a State Scholarships Foundation grant. This work has been partially supported by the European Union co-funded project Simple Mobile Services (IST-SMS, http://www.ist-sms.org/).

References

[1] AndroMDA BPM4Struts Cartridge Profile. <http://galaxy.andromda.org/docs/andromda-bpm4struts-cartridge/profile.html>.
[2] G. Bartolomeo, N. Blefari-Melazzi, G. Cortese, A. Friday, G. Prezerakos, R. Walker, S. Salsano, SMS: simplifying mobile services – for users and service providers, in: Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW 2006), IEEE Computer Society Press, 2006, pp. 209.
[3] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, T.J. Grose, Eclipse Modeling Framework, The Eclipse Series, Addison Wesley Professional, 2003.
[4] S. Ceri, P. Fraternali, A. Bongio, Web Modeling Language (WebML): a modeling language for designing Web sites, in: Proceedings of the 9th International World Wide Web Conference, North-Holland Publishing Co., Amsterdam, The Netherlands, 2000, pp. 137–157.
[5] S. Ceri, F. Daniel, M. Matera, Model-Driven Development of context-aware web applications, ACM Transactions of Internet Technology 7 (1) (2007) (Article no. 2).
[6] K. Dey, G.D. Abowd, Towards a Better Understanding of Context and Context-Awareness, Technical Report GIT-GVU-99-22, GVU Center, Georgia Institute of Technology, 1999.
[7] Eclipse Model Development Tools (MDT). <http://www.eclipse.org/modeling/mdt/?project=uml2>.
[8] Fengler, W. Fengler, V. Duridanova, Extending the modeling efficiency of the UML activity diagram for the design of distributed systems, in: Proceedings of the 2nd International Workshop on Innovative Internet Computing Systems, vol. 2346, Springer-Verlag, 2002, pp. 51–62.
[9] V. Grassi, A. Sindico, Towards model driven design of service-based context-aware applications, in: Proceedings of the International Workshop on

[10] <http://icbnet.ntua.gr/context-ws-mde>.
[11] <http://www.xmethods.com>.

Engineering of Software Services for Pervasive Environments: in Conjunction with the 6th ESEC/FSE Joint Meeting, ACM, Dubrovnik, Croatia, 2007, pp. 69–74.

[10] G. Kapitsaki, D.A. Kateros, I.E. Foukarakis, G.N. Prezerakos, D.I. Kaklamani, I.S. Venieris, Service composition: state of the art and future challenges, in: Proceedings of IST Mobile and Wireless Communications Summit, IEEE Computer Society Press, 2007, pp. 1–5.

[11] G.M. Kapitsaki, D.A. Kateros, I.S. Venieris, Architecture for provision of context-aware web applications based on web services, in: Proceedings of IEEE Personal, Indoor and Mobile Radio Communications Conference (PIMRC 2008), Cannes, France, 2008, pp. 1–5.

[12] D.A. Kateros, G.M. Kapitsaki, N.D. Tselikas, I.S. Venieris, A methodology for model driven web application composition, in: Proceedings of the IEEE International Conference on Services Computing (SCC 2008), vol. 2, IEEE Computer Society Press, Honolulu, Hawaii, USA, pp. 489–492.

[13] R. Keays, A. Rakotonirainy, Context-oriented programming, in: Proceedings of the 3rd ACM International Workshop on Data Engineering for Wireless and Mobile Access, ACM, San Diego, CA, USA, 2003, pp. 9–16.

[14] M. Keidl, A. Kemper, Towards context-aware adaptable web services, in: Proceedings of the 13th international World Wide Web Conference (WWW'04), ACM, New York, NY, USA, 2004, pp. 55–65.

[15] D. Kohlert, A. Gupta (Eds.), JAX-WS 2.1, The Java API for XML-Based Web Services, Sun Microsystems, May, 2007. <http://jcp.org/aboutJava/communityprocess/mrel/jsr224/index2.html>.

[16] S.J. Mellor, S. Kendall, A. Uhl, D. Weise, MDA Distilled, Addison Wesley Longman Publishing Co., Inc., 2004.

[17] Object Management Group (OMG), Object Constraint Language OMG Available Specification, v.2.0, 2006. <http://www.omg.org/docs/formal/06-05-01.pdf>.

[18] Object Management Group (OMG), Unified Modeling Language (OMG UML), Superstructure, v.2.1.2, 2007. <http://www.omg.org/docs/formal/07-11-02.pdf>.

[19] Object Management Group (OMG), XML Metadata Interchange (XMI), MOF 2.0/XMI Mapping, v.2.1.1, 2007. <http://www.omg.org/docs/formal/07-12-02.pdf>.

[20] G.N. Prezerakos, N.D. Tselikas, G. Cortese, Model-driven composition of context-aware web services using ContextUML and aspects, in: Proceedings of the IEEE International Conference on Web Services (ICWS'07), IEEE Computer Society Press, UT, USA, 2007, pp. 320–329.

[21] D. Schwabe, G. Rossi, Developing hypermedia applications using OOHDM, in: Proceedings of the Workshop on Hypermedia Development Processes, Methods and Models, Pittsburgh, USA, 1998.

[22] Q.Z. Sheng, B. Benatallah, ContextUML: a UML-based modeling language for model-driven development of context-aware web services, in: Proceedings of the International Conference on Mobile Business (ICMB2005), IEEE Computer Society Press, 2005, pp. 206–212.

[23] D. Skogan, R. Gronmo, I. Solheim, J. Oldevik, Model-driven web services development, in: Proceedings of the IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE2004), IEEE Computer Society Press, Taipei, Taiwan, 2004, pp. 42–45.

[24] H. Tai, K. Mitsui, T. Nerome, M. Abe, K. Ono, M. Hori, Model-driven development of large-scale Web applications, IBM Journal of Research and Development 48 (5/6) (2004) 797–809.

[25] G. Tandon, S. Ghosh, Using subject-oriented modeling to develop Jini applications, in: Proceedings of the 8th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2004), IEEE Computer Society Press, 2004, pp. 111–122.