Adaptive Web Services for Modular and Reusable Software Development:

Tactics and Solutions

Guadalupe Ortiz
University of Cádiz, Spain

Javier Cubo
University of Málaga, Spain

Information Science
REFERENCE

Managing Director: Lindsay Johnston
Senior Editorial Director: Heather A. Probst
Book Production Manager: Jennifer Romanchak
Publishing Systems Analyst: Adrienne Freeland
Managing Editor: Joel Gamon
Development Editor: Hannah Abelbeck
Assistant Acquisitions Editor: Kayla Wolfe
Typesetter: Travis Gundrum
Cover Design: Nick Newcomer

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com

Library of Congress Cataloging-in-Publication Data

Adaptive web services for modular and reusable software development: tactics and solutions / Guadalupe Ortiz and Javier Cubo, editors.
p. cm.
Includes bibliographical references and index.
Summary: “The book comprises chapters that present tactics and solutions for modular and reusable software development in the field of adaptive Web services”--Provided by publisher.
ISBN 978-1-4666-2089-6 (hardcover) -- ISBN 978-1-4666-2090-2 (ebook) -- ISBN 978-1-4666-2091-9 (print & perpetual access)
1. Web services. 2. Computer software--Reusability. 3. Component software 4. Computer software--Development. I. Ortiz, Guadalupe, 1977- II. Cubo, Javier, 1978-
TK5105.88813.A365 2012
006.7’8--dc23
2012013952

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Chapter 12
Addressing Device-Based Adaptation of Services: A Model Driven Web Service Oriented Development Approach

Achilleas P. Achilleos
University of Cyprus, Cyprus

Kun Yang
University of Essex, UK

George A. Papadopoulos
University of Cyprus, Cyprus

ABSTRACT

The rapid growth of the mobile devices market and the increasing requirements of mobile users augment the need to develop Web Service clients that could be deployed and run on both mobile and desktop devices. Different developers attempt to address this heterogeneity requirement and provide solutions that simplify and automate the development of device-aware services. This chapter proposes a Model-Driven Web Service oriented approach, which allows designing and automatically generating mobile and desktop-based clients that are able to invoke ubiquitously Web Services from different devices. This is further enabled via the Web Services Description Language that allows generating the required proxy classes, which support the communication with platform-specific clients. The applicability and efficiency of the approach is demonstrated via the design and development of a device-aware Web Service prototype.

DOI: 10.4018/978-1-4666-2089-6.ch012

INTRODUCTION

Mobile devices have obtained great prominence in the marketplace (Bartolomeo et al., 2006) and mobile users requirements have significantly increased in terms of running mobile services on these devices (Kapitsaki et al., 2009). The continuous development of existing technologies (e.g. J2ME, C#) and the introduction of brand new technologies (e.g. Android) raises new requirements and imposes new restrictions when developing service-clients (Daniel Dern, 2010). Consequently, an all-important constraint arises, which is principally associated with the interface limitations and restrictions imposed when developing platform-specific service clients for invoking and utilising Web Services from different devices.

During the early days of computing, the development of complete desktop-based applications was the main focus of developers. With the advent of Web Services the focus shifted to the development of services designed to be accessible from resource-rich (i.e. desktop, laptop) devices. Nowadays, the rapid and continuous growth of mobile devices hardware and software technologies shifted the focus towards mobile computing. Thus, the necessity arises to design Web Services in a flexible way because of the requirement to invoke them from different types of devices; i.e. mobile and stationary. This prerequisite perplexes the development of platform-specific service clients (running on different mobile devices) mainly because of interface limitations and restrictions; e.g. screen size, resource-constraints, processing power.

In this chapter we concentrate on the formulation of a model-driven approach, which attempts to exploit also the benefits of the Web Services technology. The key point is the separation of the development of the service clients from the implementation of the functionality of the Web Services. This offers a flexible, modular and abstract approach, which simplifies and accelerates the development of device-aware Web Services. The term device-aware Web Services refers to the development of both the service clients and the server-side functionality (i.e. Web Service). Hence, such an approach automates and speeds up development for the following categories of devices (Ortiz and Prado, 2009):

- **Resource-Rich Devices:** These refer to powerful desktop and laptop devices that do not impose restrictions in terms of processing power, memory, screen size, etc.
- **Resource-Competent Devices:** An intermediate category of devices that are not as powerful as the above but have higher computing resources than mobile devices and smartphones; e.g. Netbooks, IPad, Kindle.
- **Resource-Constrained Devices:** Devices such as smartphones and mobile phones that have inferior computational power, memory, interface capabilities, etc. Also, they support a restrictive set of Application Programming Interfaces (APIs).

In order to accomplish this objective, the Presentation Modelling Language (PML) is defined that allows designing and automating the implementation of service clients for the above categories of devices. Moreover, the Web Services Description Language (WSDL) is exploited since it allows designing and automatically generating the required device-specific proxy classes for each service client, which support communication with the Web Service. In this way, we allow users to design Web Service clients in the form of graphical user interfaces (GUIs) and collections of communication endpoints capable of exchanging messages (W3C, 2001) with implemented Web Service(s). Both definitions are specified in the form of graphical models that are transformed to different platform-specific implementations and deployed on the corresponding devices to enable access to the Web Service(s). Thus, an

experienced developer only requires to implement the main functionality of the Web Service (at the server-side) in one of the many possible implementations; e.g. Java, .NET.

In this work, the BookStore Web Service is manually implemented in Java. This service enables the user of a mobile or desktop device to search and retrieve information on specific books. Following, it enables the user to provide his personal and payment details to complete the purchase of the book. The service clients of this prototype device-aware Web Service are designed in the form of a single platform-independent GUI model that is subsequently transformed to various target implementation technologies (e.g. C#, Android, J2ME). In addition, the communication endpoints of the clients with the Web Service are defined in the form of a WSDL model. In particular, the model describes in an abstract form the operations and parameters of the methods implemented in the Web Service. Hence, both the PML and WSDL models are transformed to the corresponding implementation technologies to enable the communication with the Web Service. Note that the transformation of GUI models is accomplished using the code generators defined in this work, while the transformation of WSDL models is achieved via the use of existing platform-specific code generators.

The rest of the chapter is structured as follows: The following section presents related work, which motivates the research steps undertaken in this work to extend the current state of the art on device-aware Web Service development. The third section introduces the architecture of the Model-Driven Web Service oriented approach proposed in this research work. Following, an initial requirement analysis of the GUI modelling domain is performed, the PML is defined in the form of an Eclipse Modelling Framework (EMF) (EMF, 2011) metamodel and the necessary domain-specific constraints are defined and imposed onto the PML. The fifth section introduces the PML code generation process and the platform-specific code generator tools defined and used in this work. It also presents example code generation scripts and template definitions for the Android implementation technology, in order to showcase how code generators are developed for the different target implementations. The following section demonstrates the design and implementation of the BookStore device-aware Web Service prototype. Finally, conclusions and plans for extension of this research work are presented in the final section.

BACKGROUND

The development of GUIs is a difficult and essential task in software development. In particular, the development overheads are largely increased while developing the same software service for miscellaneous platforms that have different requirements and impose different restrictions (Jelinek and Slavik, 2004). This applies explicitly to mobile services since the complexity of implementing GUIs is increased due to the advanced user-service interaction and the heterogeneity requirement. Thus, the capability must be provided to define GUIs in an abstract manner, which can support the advanced interaction of the user with the software application (Sauer et al., 2006), (Heines and Schedlbauer, 2007). Moreover, generation of different implementations from the same GUI models should be feasible in order to enable the invocation of Web Services from different mobile devices. In conclusion, the MDD approach should provide the capability to generate the implementation that enables communication with the implemented Web Service(s). Thus, the developer would only require implementing the main functionality of the Web Service in a single programming language or using an existing Web Service.

One initial research work on GUI modelling focuses on the design of the structure of the user interface using presentation diagrams and its

behaviour with hierarchical statechart diagrams (Sauer et al., 2006). This design step is performed using the developed GuiBuilder modelling tool that supports also the transformation of the models to the corresponding Java-based code. Also, the GuiBuilder supports the simulation of the modelled behaviour being generated. The main objective though of this work is to develop a tool that supports the model-driven development of graphical multimedia user interfaces. As a result the approach does not deal with the development of complete applications but sticks simply to the development of Java-based GUIs for the multimedia domain. The authors do state that in future work the attempt will be to demonstrate the flexibility of the transformation approach by tailoring the generator function to other implementations; i.e. addressing application heterogeneity.

Link et al. (2008) concentrate also on the aspect of user interaction by proposing a simple, tool-supported approach for the model-driven development of graphical user interfaces for miscellaneous target platforms. The goal is to define GUI features of the application via modelling and transforming these features to source code. In particular, explicit transformation rules are defined that allow transforming platform-independent models (PIMs) to platform-specific models (PSMs). Then additional transformation rules (i.e. code generators) are defined that support the transformation of PSMs to GUI-specific source code. This provides an increased automation in software development but solely for the development of the GUIs of the application.

Balagtas-Fernandez and Hussmann (2008) take the above research work a step further since they address the development of fully functional software applications for mobile platforms; by developing PIMs of an application. Their main goal is to provide the capability to non-expert users to design specialised mobile applications with ease. In particular, the authors state that: it still takes a large amount of skill and familiarity with different APIs to create a simple mobile application. Thus, developers need to consider the different API restrictions imposed on mobile applications such as device limitations (e.g. memory, screen size, power consumption) and finally, even setting up the different development environments remains still a complex task. Hence, a modelling tool named Mobile Applications (MobiA) modeller is developed, which allows designing mobile applications that could be transformed to the corresponding platform-specific code. Note that the development of transformation tools is considered as future work and thus no documented results are reported on the capability of addressing heterogeneity. Also the approach does not exploit the potential of the Web Services technology, which means that the functionality of the application is implemented on the actual device. Thus a great burden is enforced on resource-constrained devices and interoperability still remains an open issue.

The work performed by Dunkel and Bruns (2007) also attempts to provide a simple and flexible approach for developing mobile applications. The authors acknowledge that despite significant progress in the development of mobile applications, there is still a lot of space for improvements by employing code generation and declarative approaches. Hence, a model-driven approach is presented that allows modelling the user interface of the client and the service workflow in the form of graphical models. From these models the XML-based descriptions are generated as XForms code. The XForms W3C standard was specifically selected due to its close correlation with the Mobile Information Device Profile (MIDP) of J2ME. Hence, it allows mapping XForm elements to MIDP elements and generating the corresponding source code; e.g. service-client J2ME code. This discloses that the approach is tailored to the J2ME technology, although it can be extended by defining additional code generation tools for addressing different mobile implementation technologies. Moreover, as the authors state, a drawback of the approach is that miscellaneous Unified Modelling Language (UML) (OMG UML, 2007) tools must

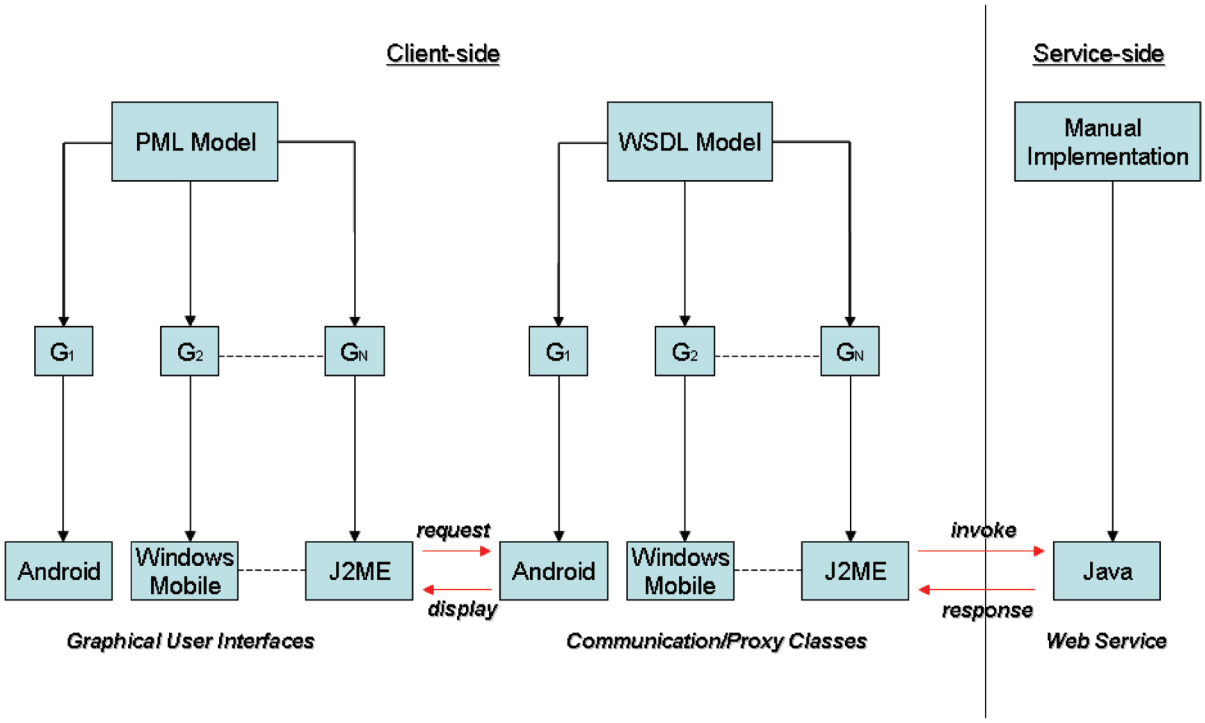
be integrated and used that do not fully support metamodelling and provide proprietary and not yet stable code generation tools.

Ortiz et al. (2009) state that device diversity and their non-stop use in everyday life activities reveals the necessity to access Web Services from these mobile devices. The main objective of their approach is to adapt the result of the Web Service invocation in accordance to the type of the client's device. Thus, they propose a server-side approach that allows developers to extend the implemented service through aspect-oriented development, so as to enable the adaptation of the result depending on the client device. Note that the Web Service code is not directly affected but rather an aspect is developed that intercepts the invocation of the service operation and adapts it according to the type of device it detects. The approach suffers though from three main issues: (i) the necessity arises to implement on the client-side the functionality that allows detecting the type of the device that invokes the Web Service, (ii) it increases response time since the aspect code must process and adapt the result in accordance to the content of the Simple Object Access Protocol (SOAP) header and (iii) it does not consider the implementation of platform-specific service clients (i.e. GUIs); authors state that GUI restrictions imposed by mobile platforms increase the complexity of developing clients (Ortiz & Prado, 2009).

MODEL-DRIVEN WEB SERVICE ORIENTED ARCHITECTURE

The proposed approach combines the Model Driven Architecture (MDA) paradigm (Kleppe et al., 2003; Singh and Sood, 2009) and Web Services technology in an attempt to exploit their potentials and overcome their limitations. Figure 1 presents the architecture of the MDD approach that comprises the client-side and the service-side.

Figure 1. Model-driven Web service oriented architecture



The client-side refers to the graphical user interfaces deployed on the mobile or desktop device, which allow the user to interact with the service by exchanging the necessary information. In order to accomplish this interaction the required communication classes (i.e. proxy classes) are used that act as the client-side connection endpoints. These end-points allow invoking Web Service(s) operations and retrieving responses through the exchange of SOAP messages.

Both the GUIs and the communication classes are automatically generated from the abstract models (i.e. PML, WSDL); as shown in Figure 1. In particular, different code generators (G1, G2, ..., GN) are defined in this work that allow transforming the PML model to different target implementations; e.g. Android, Windows Mobile. Also, existing code generation tools are utilised that support the transformation of the WSDL model to platform-specific implementations. For instance, in the case of the Android platform, the Android GUI classes utilise the Android proxy classes to invoke the Web Service, receive the appropriate response and display the information on the corresponding Android mobile device. The aforesaid communication method described for the Android platform is valid for the rest of the implementation technologies.

On the service-side the Web Service functionality is the only part that requires to be implemented manually by developers. The service functionality is coded though in one implementation technology (in this case the implementation is Java-based) and can be consumed using the capabilities of the Web Services technology by different clients. These clients do not need to be implemented using the same implementation platform (i.e. Java) but can be coded in different target implementations. This is possible since the communication is performed using SOAP, which is a simple protocol specification for exchanging XML-based structured information in computer networks. It also relies on Remote Procedure Call (RPC) and HyperText Transfer Protocol (HTTP) protocols for connection negotiation and message transmission. Thus, a service-client implemented in .NET or C# (i.e. Windows mobile) is able to communicate with the Java-based Web Service deployed and executed on a GlassFish Axis Web Server. This enables as a result interoperability, which is a widely-accepted and proven characteristic of Web Services (Kapitsaki et al., 2008), between the different platforms and simplifies and enables the rapid development of fully functional device-aware Web Services.

THE PRESENTATION MODELLING FRAMEWORK

Requirement Analysis

Service heterogeneity refers to the capability to deploy the same software service (i.e. application) on different devices. In this work service heterogeneity is satisfied via a model-driven, Web Service oriented approach that allows designing the different artefacts of the application in the form of models. Therefore, from the models (i.e. GUI, WSDL) the source code is generated for different platform-specific implementations. As aforesaid the implementation includes the service-client GUIs and the proxy classes that enable the communication with the Web Service. In particular, the generation of the source code that implements the graphical user interfaces for the client-side is very important. This aspect is fundamental due to the difficulties imposed on developers when implementing GUIs, due to the restrictions and limitations that each technology imposes. Thus, by automating the implementation of GUIs for the service-clients the development of the complete device-aware Web Service is simplified and expedited.

In addition to service heterogeneity the interaction of the user with mobile services should be simplified by providing easy-to-use and highly-capable graphical user interfaces. Due to user's mobility and the diversity of devices that need

to run mobile services, the requirement arises to design GUIs in an abstract manner that allows generating the different implementation for this assortment of devices. This ensures as a result that there is no compromise in terms of developing GUIs and thus the interaction of the user with the service remains quite straightforward but still of the highest quality.

To gratify these requirements it is imperative to provide a flexible and extensible model-driven approach that allows designing GUIs and generating the implementation code using model-to-text transformations. Hence, the Presentation Modelling Language must include abstract concepts that can be mapped using the necessary transformation rules to different implementations. For example, the top element component for implementing GUIs in Java is provided either using the *javax.swing.JFrame* or the *java.awt.Frame* implementation classes. Similarly, the *javax.microedition.lcdui.Display* class is the top-level component of the J2ME technology, the *android.app.Activity* class is the top-level of the Android technology and the *System.Windows.Forms.Form* class is the top level of the Windows Desktop and Mobile technologies. Thus, we define a modelling element of the PML that represents in an abstract form any of the above top level components without considering the final implementation technology. Furthermore, we define abstract properties for this element that can be mapped using transformation rules to the respective graphical properties of each implementation technology.

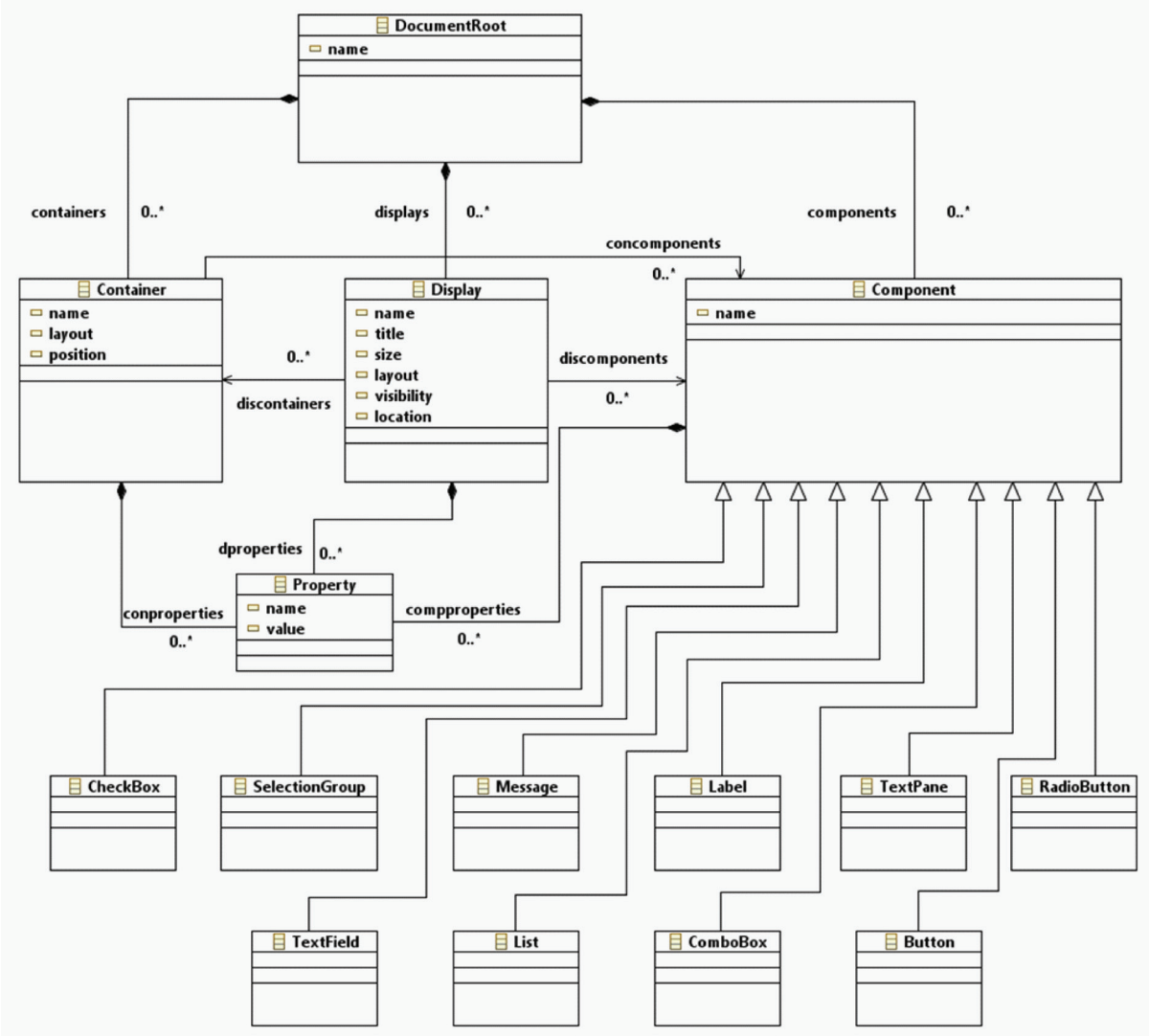
Apart from the aforementioned components, the most important and widely used components of the implementation technologies are identified and included in the form of abstract modelling elements in the PML metamodel. In particular, the platform-specific components selected, are the ones that represent equivalent/comparable graphical concepts in all the implementation technologies. For instance, the *java.awt.Label*, the *javax.microedition.lcdui.StringItem* and the *android.widget.TextView* classes serve a similar purpose, since they all represent a component capable of displaying a single-line of read-only text. Another example is the graphical component that serves the task of editing a single line of input text, which is implemented in the Java technology via the *java.awt.TextField* or *javax.swing.JTextField* class, in the J2ME technology via the *javax.microedition.lcdui.TextField* class and in the Android technology platform via the *android.widget.EditText* class. Thus (considering all cases) the abstract representation of the platform-specific graphical components is deducted and defined accordingly in the PML. Moreover, an additional requirement is captured in the PML definition that refers to the capability to represent the relationships between displays, containers and secondary components.

Finally the capability is provided to define different graphical properties for each of the above components. The different restrictions and limitations imposed by the different platforms call for a flexible and extensible approach when defining graphical properties. This provides the capability to extend the PML definition and introduce new properties, which may be added in any of the implementation technologies addressed in this work. The only prerequisite is that the transformation of these properties to implementation code is supported by the transformation rules defined within the code generators. Also the coherency of the PML definition (in terms of the many different properties that can be defined) is preserved by imposing and enforcing the necessary Object Constraint Language (OCL) (OMG OCL, 2006) rules. These OCL constraints ensure that only the permitted graphical properties can be specified in the PML model definition.

Presentation Modelling Language

The requirements analysis presented in the previous subsection drives the derivation and definition of the GUI concepts in the Presentation Modelling Language. In this work the PML is defined in the form of an EMF-based metamodel using the

Figure 2. EMF metamodel definition of the presentation modelling language



Graphical Modelling Framework (GMF) Ecore diagram tool included in the MDD environment presented in our previous work (Achilleos et al., 2007; Achilleos et al., 2008). Figure 2 illustrates the PML metamodel that defines the modelling elements, associations and properties that support the design of PML models; i.e. abstract notions of graphical user interfaces.

The top-level modelling element defines the PML model and is specified in the PML definition using the *DocumentRoot* metaclass. From the root metaclass aggregations are defined that represent the containment relationships with the rest of the modelling elements of the GUI model. Initially, the displays aggregation showcases that each PML model may contain different displays (i.e. screen of a mobile device). In order to reduce though the complexity of a PML model an OCL constraint is defined that restricts the definition to a single display within each model. This improves also the comprehension of a PML model and avoids designing large and complex PML models. Each

display element is defined as an instance of the *Display* metaclass, which includes common graphical properties such as *name, title, size, layout, visibility and location.* The first property defines the element's name, while the second property defines the title that will be displayed on the frame. In addition, the size and layout properties refer respectively to the size of the frame and the layout of containers on the frame. Finally the visibility property defines if the frame is visible or not and the location specifies the actual position of the frame on the device's screen.

Some of these graphical properties are essential for some platforms (i.e. Java) but not for others (i.e. J2ME). For instance, the layout and size properties are not necessary for J2ME code generation since a default layout manager handles the size and positioning of components on the display. Following, the containers aggregation defines that each PML model can include one or more container components. These components are described as the carriers of secondary graphical components (e.g. labels, textfields), which are represented for example in Java using the *javax.swing.JPanel* API class and in J2ME using the *javax.microedition.lcdui.Form* API class. Each container element includes also the *name, layout and position* properties that control the look-and-feel of containers. Once again these properties could be required by some implementation platforms, while for others may not be as important due to the use of layout managers.

Apart from the single display and its containers, the PML definition includes also miscellaneous important graphical components that are common and widely-used in all implementation platforms. Foremost the *Component* abstract metaclass defines the superclass of the subclasses: *Message, Label, Button, TextPane, RadioButton, ComboBox, CheckBox, TextField, List and SelectionGroup*. By creating instances of these metaclasses the designer is able to model different secondary graphical components (e.g. *Android TextView, J2ME StringItem, Java JLabel*) using an abstract representation. For instance, the *TextPane* metaclass represents input text-components that allow to display and/or edit large text. In the case of the Java abstract *TextPane* elements are represented by *javax.swing.JTextPane* API class, while in the case of Windows they are represented by the *System.Windows.Forms.TextBox* API class.

The *discontainers*, *discomponents* and *concomponents* associations complement the aforementioned modelling elements since they define the relationships between them. Foremost the *discontainers* association represents the containment relationship of the display element with one or more container modelling elements. In the case of J2ME, this corresponds to the containment relationship of the MIDlet display component with its Form container components. Next, the *discomponents* association defines a containment relationship of the display element with one or more secondary containers. Finally, the *concomponents* association defines that each container component may include one or more secondary components. For instance, in the case of Java technology, the association represents the containment of secondary graphical components (e.g. JLabel) within container components (e.g. JPanel).

The last element defined in the PML metamodel is the *Property* metaclass. This modelling element allows defining different graphical properties for display, container and component modelling elements. In fact, the aggregation associations named *dproperties*, *conproperties* and *compproperties* define clearly that each of the aforesaid modelling elements may contain one or more graphical properties defined as an instance of the *Property* metaclass. These properties are defined using the *name* attribute in the form of keywords and can be parsed via the transformation rules defined in the code generators. Furthermore, using a single OCL constraint we are able to ensure that the designer will be permitted to define keywords (i.e. graphical properties) that are supported in the current version of the PML. This provides

the flexibility and extensibility to add/remove new properties by adding/removing keywords to this OCL rule. Finally the *value* attribute defines the value of the *Property* element, which may represent (for example) the actual text that will appear on a label.

Presentation Modelling Language Constraints Definition

The definition of the PML as an EMF-metamodel allows the design PML models that represent graphical user interfaces. It is not possible to ensure the design of coherent PML models via the metamodel definition. Thus, it is important to define the domain-specific modelling rules that restrict the definition of PML models to valid graphical user interfaces. This provides a complete and coherent PML definition and allows generating the corresponding Presentation Modelling Framework (PMF), using the capabilities of the generic MDD environment (Achilleos et al., 2007; Achilleos et al., 2008). The PMF comprises a modelling editor with drag-and-drop capabilities that permits the design and validation of PML models.

From the complete set of OCL rules imposed to the PML the following two are selected to showcase their importance and reveal also how the flexibility and extensibility of the PML is preserved. The primary OCL constraint is imposed onto the *Display* metaclass in order to restrict the definition of the container's position property in accordance to the layout property of the display. In particular, the constraint defines that in the case that the layout property is set as *"default"* the position property of the associated containers should be defined using the following values: (i) *CENTER*, (ii) *EAST*, (iii) *WEST*, (iv) *NORTH* and (v) *SOUTH*. For instance, in the case of the Java platform the *"default"* value defines that the corresponding *BorderLayout* API class should be generated and used. This class permits placing the display's containers in one of the above positions; e.g. CENTER. Furthermore, the *"else"* conditional statement specifies that the position property should be defined as an *Integer* since the current layout refers to the *GridLayout* API class. The integer indicates as a result the index number that reveals the position of the container. Note that in the current version of the PML only the *BorderLayout* and *GridLayout* managers are supported for the design and transformation of GUI models to Java-based code.

```
context Display
inv: if self.layout = 'default'
then self.discontainers→forAll(con:
Container | con.position = 'CENTER')
or self.discontainers→forAll(con:
Container | con.position = 'EAST')
or self.discontainers→forAll(con:
Container | con.position = 'WEST')
or self.discontainers→forAll(con:
Container | con.position = 'NORTH')
or self.discontainers→forAll(con:
Container | con.position = 'SOUTH')
else self.discontainers→forAll(con:
Container | con.position.toInteger().
oclIsTypeOf(Integer))
endif
```

The second OCL constraint presented in this work showcases the importance of OCL rules and reveals also how the flexibility and extensibility of the PML is preserved. This domain rule defines the keywords that are currently supported by the PML and can be defined as graphical properties of the PML elements. In the case the designer attempts to assign a non-supported property to any modelling element then a constraint violation is raised that reveals the design error. In this way the capability is provided to extend the PML definition simply by adding new keywords to the following OCL constraint. Thus, the flexibility and extensibility of the PML is preserved, providing

also the capability to address the miscellaneous graphical requirements and restrictions that different platform-specific implementations impose. The definition of OCL constraints completes the PML definition and enables the design and validation of PML models.

```
context Property
inv:Property.allInstances()→forAll(p:
Property | p.name = 'text' or p.name
= 'title' or p.name =
'message' or p.name = 'rows' or
p.name = 'columns' or p.name = 'line-
Wrap' or p.name =
'stringArray' or p.name = 'command')
```

PLATFORM-SPECIFIC CODE GENERATORS DEFINITION

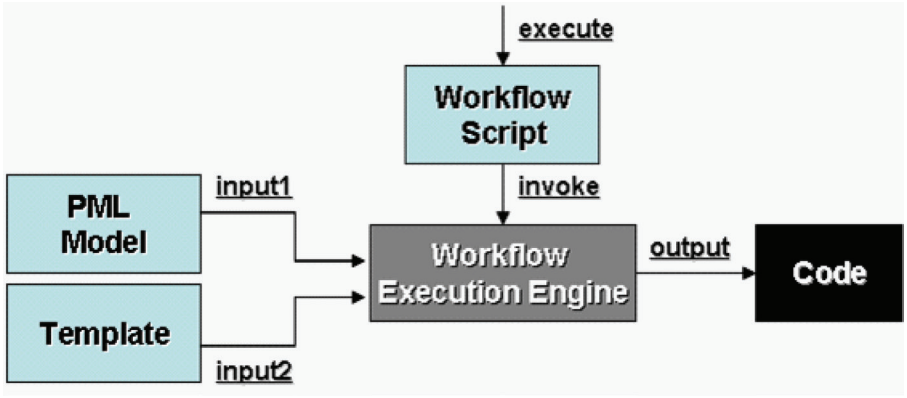
In order to support the transformation of PML models to the necessary platform-specific implementations it is required to define precise transformation rules that compose the code generators. The definition of precise transformation rules is imperative, in order to ensure the correctness of the operational semantics of the generated implementation. Therefore, apart from the necessity to design precise PML models it is required to exploit a widely-used MDD approach that simplifies and aids the definition of coherent platform-specific code generators. Moreover, existing WSDL code generation tools are utilised in this work to generate the service-client proxy classes that enable communication of the service clients with the Web Service.

Presentation Modelling Language Code Generation

The part of the Web Service stubs generation is not addressed in the current work. In the literature existing works on generation from WSDL descriptions exist and are employed in this work. WSDL, serving as the specification descriptor language for WSs, offers an abstract layer depicting the service functionality. Clients that wish to consume specific WSs rely on this WSDL specification in order to discover the operations supported, the input arguments needed and the expected response messages. WSDL provides a generic description independently of the network protocols that can be adopted for communication purposes. WSDL code generators can be found in Java WS frameworks, such as the Novell exteNd Director Development environment and the Axis2 Service Archive Generator Wizard offering the wsdl2java tool. .NET offers its own custom wsdl code generation tool. In the proposed framework the two latter tools have been employed along with the J2ME generator that forms part of the Sun Java Wireless Toolkit for CLDC. However, since no such tool is available for the Android platform, in the current stage of the presented work the WS communication classes were developed manually. Further discussion on code generators for WSs is not included in the current work, which focuses on the applicability of the presentation code generation tools on multi-platform environments. However it should be noted that in the framework of MDE some works exploit WS models and introduce tools for model transformation procedures. For further information on such issues the reader can refer to relevant publications (Kapitsaki et al., 2009, Gronmo et al., 2004, Sheng et al., 2005).

In terms of the presentation layer, the code generation process allows transforming PML models to the appropriate platform-specific code. A set of generators targeting various platforms of stationary and mobile devices have been implemented: Java, J2ME, Android, Windows Mobile and Windows Desktop. In this subsection the Android specific code generator is described in order to showcase the flexibility of the code generation approach. To keep the chapter comprehensive and due to space limitations it is not possible to describe the whole generators set.

Figure 3. The PML code generation process



The MDD environment (Achilleos et al., 2007; Achilleos et al., 2008) proposed in previous work features the openArchitectureWare (oAW) software tool. oAW is a MDA generator framework implemented in Java, which supports explicitly the definition of model-to-text transformation rules. It includes the Xpand template language, a template text-editor and the workflow execution engine. Also it features additional languages, namely Check and Xtend, which include their individual text-editors. Foremost, the Xpand template language supports the definition of advanced code generators in the form of templates that capture the transformation rules and control the output document generation; e.g. XML, Java, C#, HTML. These templates are defined using the Xpand text-editor and include references to extension functions defined using the Xtend language. Extension functions are considered as utility functions (i.e. similarly to Java utility functions) that support the definition of well-formulated generators and improve the structure of the generated code. Moreover, the Check language supports the definition of additional constraints using a proprietary language. Finally the workflow execution engine drives the code generation in accordance to the defined templates.

Figure 3 illustrates the code generation process that is driven by the workflow engine. The executable workflow script presented in Listing 1 allows delegating calls to the necessary Java-based classes of the oAW component. It creates an "XmiParser" component and calls the "org.openarchitectureware.emf.XmiReader" that allows loading and parsing the model into memory. In fact, the PML model ("WebServiceClients.pres") is serialised in the form of XML-Metadata Interchange (XMI) format. The metamodel defined in this work is referenced also in the script, so as to be able to recognise, parse and load elements, associations and properties defined in the PML model; i.e. making them accessible at runtime. Following, the important *"Generator"* component is defined, by referencing the *"org.openarchitectureware.xpand2.Generator"* class. Also the flag *skipOnErrors= "true"* allows terminating code generation if errors are detected in the template definition. Moreover, the component defines that the input PML model is an instance of an EMF-based metamodel (i.e. PML metamodel) and that the template definition is based on the PML metamodel.

The most important artefact in the workflow script is the template definition, which describes the rules for transforming the PML model to the corresponding code. Listing 2 presents a sample part of the Android template definition that allows demonstrating how code generation is achieved.

Listing 1. PML code generation workflow script

```
1. <workflow>
2. <component id=”xmiParser” class=”org.openarchitectureware.emf.XmiReader”>
3. <modelFile value=”models/WebServiceClients.pres”/>
4. <metaModelPackage value=”presentation.PresentationPackage”/>
5. <outputSlot value=”model”/>
6. <firstElementOnly value=”true”/>
7. </component>
.............
10. <component id=”generator” class=”org.openarchitectureware.xpand2.Generator”
11. skipOnErrors=”true”>
12. <metaModel id= “mm” class=”org.openarchitectureware.type.emf.EmfMetaModel”>
13. <metaModelPackage value=”presentation.PresentationPackage”/>
14. </metaModel>
15. <expand value=”templates::AndroidPresentation::Root FOR model”/>
.............
18. </component>
19. </workflow>
```

Listing 2. Part of the Android template definition

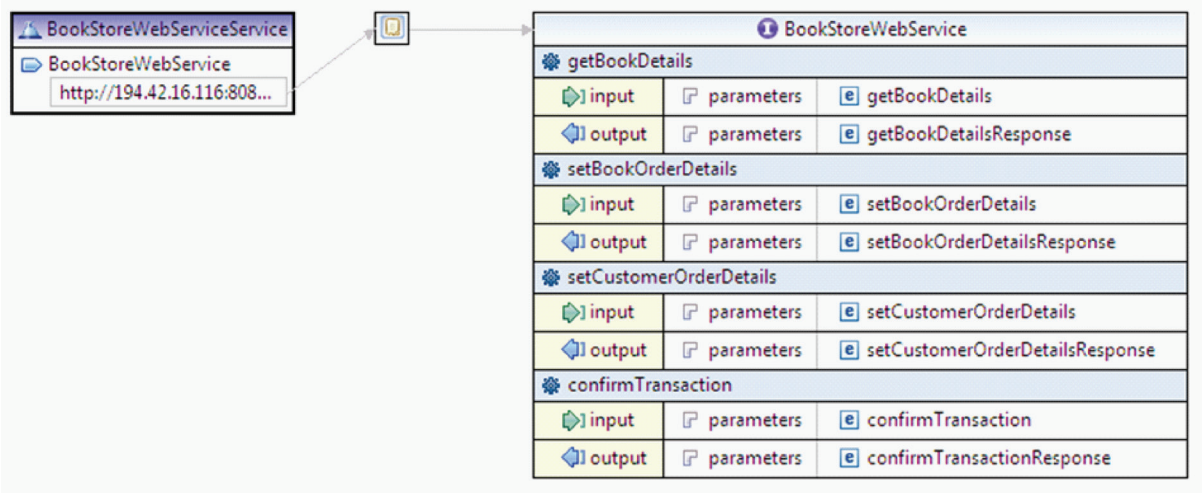
```
1.  <<EXTENSION templates::AndroidPresentation>>
2.  <<DEFINE Root FOR presentation::DocumentRoot>>
    .....
30.  <<REM>>Starts iteration and creates a View for each container.<<ENDREM>>
31.  <<FOREACH this.discontainers AS discon->>
32.  public View <<discon.name+”View”>>(){
33.  this.setTitle(<<discon.conproperties.select(e|e.name.contains(“title”)).
           value.first()>>);
34.  <<discon.name>> = new TableLayout(this);
35.  <<REM>>Create the respective components contained in each View.<<ENDREM>>
36.  <<FOREACH discon.concomponents AS concomp->>
37.  <<IF concomp.metaType.name.matches(“presentation::Label”)->>
38.  <<concomp.name>> = new TextView(this);
39.  <<concomp.name>>.setText(<<concomp.compproperties.select
                (e|e.name.contains(“text”)).value.first()>>);
40.  <<ELSEIF concomp.metaType.name.matches(“presentation::TextField”)->>
41.  <<concomp.name>> = new EditText(this);
    .....
71.  <<REM>>Ends the loop associated with the components collection.<<ENDREM>>
72.  <<ENDFOREACH>>
73.  <<REM>>Ends the loop associated with the containers collection.<<ENDREM>>
74.  <<ENDFOREACH>>
```

The main part of the sample generator presented in this work is included in lines 31-74. This part is repeated for all display containers of the model enabling access to the graphical properties of the containers and the secondary components associated to them. For instance, line 34 illustrates how we can generate an Android *TableLayout* object and set accordingly its name in accordance to the name of the current container in the iteration, i.e. << *discon:name* >>. The iteration through the collection of secondary components associated with each container is performed in lines 36-72. Depending on the type of element read and parsed (indicated by the properties of *concomp*) the respective object is generated. For instance a *TextView* object is generated for each *Label* element as indicated in lines 37-39, where the keyword "*text*" used at line 39 provides the capability to set the text on the label to the value parsed from the *Label* element definition. The list of conditional statements allows reading, parsing and generating other types of secondary components using the same reasoning. An equivalent approach was followed for defining the templates that support the transformation of the models to the other four platform-specific implementations; e.g. Java, J2ME, Windows Mobile and Desktop.

Web Service Description Language Code Generation

In the previous subsection the code generation process was explained, which allows transforming PML models and generating the service clients. This subsection introduces briefly the transformation of WSDL models (see Figure 4) to the corresponding proxy classes that support the communication with the Web Service. The WSDL serves as a common specification language for the Web Services domain, which allows defining the Web Service functionality using abstract models. Therefore, different implementation technologies have developed their own code generation tools that allow transforming these abstract WSDL models to the respective implementation classes that permit to invoke and retrieve responses from the Web Service. In this work we exploit existing WSDL code generation tools (e.g. *Axis2 wsdl2java* tool, *.NET wsdl* tool) and refrain from applying a similar code generation process such as the one described in the previous subsection. Details on the implementation of the WSDL code generation tools are out of the scope of this work.

Figure 4. The BookStore Web service description language model



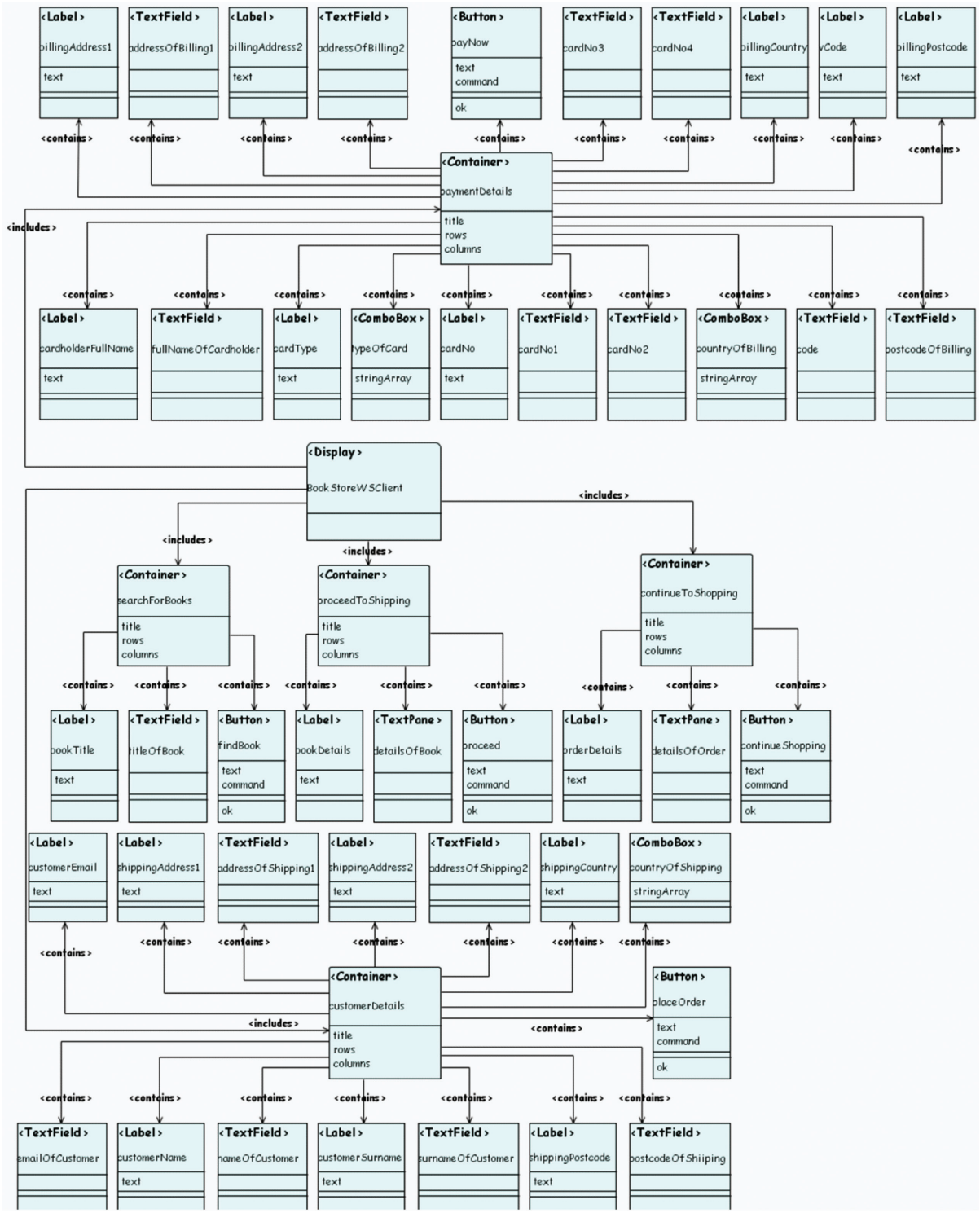
THE BOOKSTORE WEBSERVICE PROTOTYPE

The prototype is a *BookStore* Web Service that allows a user to search, find and purchase books. In particular, the user enters the necessary information on the book (i.e. book title) and invokes through a button generated event the function "getBookDetails" of the Web Service; see Figure 4. This function accepts as input parameter the title of the book and returns as output parameter the details of the book (e.g. book description, book price). The user is then able to purchase the book by invoking the "setBookOrderDetails" operation of the Web Service that stores the necessary details of the book order into the database. The next two screens allow the user to enter personal and payment information to confirm the transaction. This is performed by invoking the corresponding functions of the Web Service through user-generated events, which allow performing the necessary operations. The final screen displays to the user details on the purchase and the user may continue shopping or choose to terminate the application. Note that the functionality of the *BookStore* Web Service is implemented manually and utilises Java Open Database Connectivity (ODBC) that allows accessing database management systems (DBMS) and querying and retrieving data from the database. Moreover, different service-clients are generated by transforming automatically the PML model to different implementations and the WSDL model to proxy classes that enable communication with the Web Service.

Figure 4 presents the WSDL model that represents the abstract functionality of the *BookStore* Web Service. This model is defined using the Eclipse WSDL modelling editor plug-in integrated in the MDD environment used in this work. The model defines the abstract functions of the Web Service and their input and output parameters. As aforementioned, the necessary proxy classes are automatically generated from the WSDL model using existing code generation tools. These classes enable the communication with the Web Service by allowing the invocation of the implemented operations using the SOAP protocol, which allows exchanging messages represented as structured XML-based information. In particular, connection negotiation and message transmission are performed through the network using the RPC and HTTP protocols. In this work the functionality of the Web Service is implemented in Java and it is deployed and executed on a GlassFish Axis web server. Consequently, the different service-clients (e.g. Android, Windows mobile, and J2ME) generated from the PML model are able to invoke and utilise the functionality of the Web Service using the corresponding proxy classes.

Figure 5 illustrates the designed PML model that represents the GUIs of the service clients and its actually an instance of the *DocumentRoot* metaclass. The model defines at the center an instance of the *Display* metaclass, which is the main screen of the service. The display element is associated with four different container components that represent the different views of the service during its execution. They are defined as instances of the *Container* metaclass. The primary container (i.e. *searchForBooks*) is associated with secondary components that represent the GUI components that allow searching for a book by entering the title and invoking the corresponding Web Service operation. The second container (i.e. *proceedToShipping*) is the GUI view that includes the graphical components that allow displaying details of the book in case it is available. At this stage the user is able to invoke the Web Service function, which allows storing the information of the book into the database as part of the order details. The user is then presented with the third container (i.e. *customerDetails*), which allows entering personal and shipping information. Following, the user-generated event invokes the service function that stores this information in the database as additional order details. Finally

Figure 5. The bookstore presentation modelling language model



the user is presented with the next container (i.e. *paymentDetails*) that allows filling in the required details for completing the book purchase.

In terms of graphical representation the model is rather complex in the current version of the PML language. It can be easily adapted though so as to permit a simpler graphical representation of the elements, their associations and properties. The key point addressed in this work is the capability provided to automate the implementation of the Web Service clients by transforming the PML model to the necessary GUI implementations. Listing 3 illustrates part of the code generated from the transformation of the PML model to the Android implementation. In particular, two functions are displayed in Listing 3 that refer to the functionality that enables searching for a book. For instance, lines 1-20 are automatically gener-

Listing 3. The GUI code generated for the Android target platform

```
1.   public View searchForBooksView() {
2.              this.setTitle("BookStore - Multi-platform Web Service");
3.              searchForBooks = new TableLayout(this);
4.              bookTitle = new TextView(this);
5.              bookTitle.setText("Enter Book Title:");
6.              titleOfBook = new EditText(this);
7.              findBook = new Button(this);
8.              findBook.setText("Find Book");
9.              findBook.setTextSize(10.0f);
10.             findBook.setTextColor(Color.rgb(100, 200, 200));
11.             findBook.setOnClickListener(this);
12.             searchForBooks.addView(bookTitle);
13.             searchForBooks.addView(titleOfBook);
14.             searchForBooks.addView(findBook);
15.             /*PROTECTED REGION ID(searchForBooksAddToView) ENABLED START*/
16.             /** TODO starts */
17.             /** TODO ends */
18.             /*PROTECTED REGION END*/
19.             return searchForBooks;
20.     }
   .....
442.        /**  Called when a user event is generated.*/
443.          public void onClick(View event) {
444.                if (event.equals(findBook)) {
445.                /*PROTECTED REGION ID(findBook) ENABLED START*/
446.                /** TODO starts */
447.                proxy_stub = new AndroidServerProxy();
448.                try {
449.                _book_Details =
     proxy_stub.getBookDetails(titleOfBook.getText().toString());
        .....
```

Figure 6. The bookstore Web service deployed on different platforms

ated via the execution of the transformation rules defined in lines 30-74 of Listing 2. These rules are applied on the *Container, Label, TextField and Button* elements of the *searchForBooks* container illustrated in Figure 5. The developer must implement manually a few lines of code (i.e. protected TODO branches), which handle adding components on the container and invoking the appropriate Web Service function (lines 445-449) using the proxy classes generated from the WSDL model. Therefore, the implementation effort is significantly reduced since a large percentage of the GUI code is generated from the PML model. Equivalent transformation rules are defined and applied for generating the GUI implementation for Java, J2ME, Windows Mobile and Desktop platforms. Therefore Web Service heterogeneity is achieved via the automatic generation of the service-clients GUI implementation for miscellaneous platforms.

Figure 6 demonstrates screenshots captured during the use of the *BookStore* Web Service on mobile clients deployed on the Android and Windows Mobile platforms. The screens for searching for a book, displaying the results and filling out the information for purchasing the book are displayed in the figure. Alternated screenshots capture different steps during the execution of the service on these platforms. Moreover, a brief evaluation of our approach using the LoC software metric is performed. Our attempt is to showcase the reduction of the coding effort when developing the *BookStore* Web Service for the platforms presented in our case study.

Table 1 presents the results obtained by comparing the code generated from the models (i.e. PML, WSDL) against the full implementation code for each platform-specific service client. It

Table 1. Evaluation results on the model-driven, Web Service-oriented approach

LoC Metric (per platform)	Generated Code	Overall Code	Generated/ Overall (%)
Java	189	334	56.59
J2ME	267	369	72.39
Android	244	361	67.59
Windows Mobile	360	481	74.84
Windows Desktop	360	475	70.3
All Platforms	1420	2020	70.3

is important to point out that the Web Service functionality is implemented only once using Java and is consumed by different clients. Therefore, the Web Service functionality implementation (i.e. 55 lines of code) is considered when deriving the percentage for all the target platforms. As can be observed from the results a significant part of the clients' code has been generated for different platforms; i.e. percentages are well above 50%. Moreover, our experience in defining transformation rules for the Java and J2ME platform (Achilleos et al., 2009) suggests that code generators can be further optimised in order to achieve higher-degree of automation. For instance, the generation percentage for Java is significantly lower since the code for placing components on containers must be manually implemented. In all other platforms a default layout manager handles placing components on the container. Thus, for those platforms it is only necessary to add the components to the container. Finally, the number of platforms considered in this work showcases the flexibility and applicability of the transformation method so as to be applied successfully to other platforms; e.g. Nokia Symbian OS, Apple iOS, BlackBerry RIM.

COMPARISON WITH EXISTING WORK

In terms of our approach we focus on the following requirements for simplifying and expediting the development and deployment of device-aware Web Services. These are: (i) the degree of automation in service development and (ii) device heterogeneity. Additional requirements considered have to do mainly with the Web Services technology. These were extracted from the study of existing work that deals with the development of Web Service-oriented device-aware applications. These requirements are namely: (i) service interoperability, (ii) service transparency, (iii) service consistency, (iv) code duplicity and (v) user-awareness (Ortiz & Prado, 2009). Table 2 illustrates a comparison with existing work, which focuses on two development aspects. At first, research work focuses on model-driven development of graphical user interfaces, so as to simplify and accelerate the development of complete mobile applications. More recent research work focuses on simplifying and automating the development of device-aware Web Services using either model-driven and/or aspect-oriented approaches.

The initial research effort conducted by Sauer et al. (2006) addresses the model-driven devel-

Table 2. Comparative analysis of MDD approaches for device-aware Web services

	Main requirements		**Web Service Requirements**				
	Development Automation	Heterogeneity	Interoperability	Service-side Transparency	Service Consistency	Non-duplicity	User-awareness
Sauer et al.	*M/H*	*M/H*	*NA*	*NA*	*NA*	*NA*	*NA*
Link et al.	*H*	*H*	*NA*	*NA*	*NA*	*NA*	*NA*
Balagtas & Hussmann	*L*	*L*	*NA*	*NA*	*NA*	*NA*	*NA*
Dunkel & Bruns	*H*	*H*	*H*	+	+	*x*	*x*
Ortiz et al.	*M*	*H*	*VH*	+	*x*	+	*x*
Our approach	*VH*	*VH*	*VH*	+	+	*x*	*x*
	L: Low, M: Medium, H: High, VH: Very High *NA: Not Applicable, +: Satisfied, x: Not Satisfied*						

opment of GUIs in an attempt to simplify the development of multimedia applications. The authors have build a prototype GUI modelling tool (i.e. GuiBuilder) that allows to design the structure of a multimedia user interface using presentation diagrams and its behaviour with hierarchical statechart diagrams. This initial work demonstrates the potential of modelling multimedia GUIs and generating automatically the Java SWT implementation for the multimedia application. The approach though does not reveal how transformation rules can be tailored, so as to address the heterogeneity requirement. Also, the modelling tool is manually implemented using the Plug-in Development Toolkit (PDT) and the Graphical Editor Framework (GEF) of Eclipse. In this work we claim that a model-driven development approach should allow generating the modelling tools being used. Thus, a fully integrated MDD environment is preferred that provides the capability to generate the modelling tools and allows to easily define transformation rules that target different implementations. Moreover, such an approach allows extending/modifying the modelling language and regenerating its modelling tool. Finally this approach does not deal with the Web Services technology.

A similar approach is defined by Link et al. (2008) that concentrates on the aspect of the interaction of the user with the application. Their objective is to define GUIs in the form of models and transform these models into source code; i.e. targeting miscellaneous platforms. The proposed MDD approach defines in fact two UML profiles that support the model-driven development of GUIs and specifies transformation rules using the Query/View/Transformation (QVT) standard. Hence, the approach complies largely with the MDA paradigm and provides as a result general applicability and flexibility in terms of modelling and definition of transformation rules. Therefore, the degree of automation in software generation is considerably high and the capability is also provided to easily define transformation rules for miscellaneous platforms. The only predicament is that UML tools do not satisfactorily support metamodelling and do not provide highly-competent and stable code generation tools. Once again the approach tackles merely the development of GUIs, although it can be tailored to address also the development of complete Web Service-based applications.

The approach proposed by Balagtas-Fernandez and Hussmann (2008) considers the model-driven development of fully functional mobile applications rather than just the GUIs of the application. This preliminary research work developed an initial modelling prototype tool that allows defining a user interface model that describes the GUIs, a navigation model that defines how the mobile application navigates from one screen to the next and the information retrieval model that helps in showing how information is exchanged between models. The development of the necessary transformations rules is not a main focus of this work. Authors do state though that in future work the objective is to provide rules that transform the graphical models to XML-based models and then to code. Consequently, the merits of the approach in terms of development automation and application heterogeneity are still to be proven. Moreover, the approach examines the development of complete applications that run on the mobile device, which is not highly-suitable for resource-constrained mobile devices.

Dunkel and Bruns (2007) declare that a powerful architecture is indispensable for applying model-driven development of mobile applications and achieving automation and heterogeneity. The authors propose the BAMOS platform that comprises different architectural components: (i) Service Provider - offers implemented services (i.e. Web Services) to other systems, (ii) Service Broker - acts as the mediator between Service Providers and Adhoc Clients and (iii) Adhoc Client - software component running on a mobile device. Hence, the BAMOS architecture provides the necessary interoperability and allows an Adhoc Client

to use different services; e.g. Web Services. The approach provides a Domain Specific Language (DSL) (Dunkel & Burns, 2009) defined as a UML profile (similar to a metamodel) for developing mobile applications based on BAMOS. The defined models describe the Adhoc client (i.e. GUIs) and the service work flow specification that can be transformed to XForms code. Subsequently, the XForms representation can be transformed to J2ME code due to its strong correlation with MIDP. Thus the approach is bound to the J2ME implementation and the BAMOS platform, although it can be adapted to target other platforms.

The architecture of the approach, i.e. being Web Service oriented, allows evaluating the approach against the Web Service requirements. First the approach satisfies service-side transparency because the Web Service does not need to implement complex code, which allows detecting from which device the service is invoked. A simple option will be to duplicate methods in the Web Service and thus each client may invoke a different method according to the device that the client is deployed. This implies though that some code is duplicated in the Web Service implementation. Also service consistency is preserved since the generated service implementation is consistent with the original service definition; i.e. no additional code needs to be inserted at the service-side. In addition, the approach does not consider user-awareness, which means that the user is not able to intervene and adapt the service response in accordance to his/her preferences; e.g. displaying fewer output information.

One of the most competent approaches (especially in terms of satisfying Web Service requirements) is the model-driven, aspect oriented approach proposed by Ortiz et al. (2009). The authors describe different techniques for adapting Web Services for different devices and choose from these alternatives the model-driven, aspect-oriented technique. This technique allows adding an optional tag in the SOAP message header so as to adapt the results in accordance to each device. The technique is characterised by service transparency and non-duplicity of code at the service-side. It does not satisfy though service consistency since the SOAP header is modified and through the adaptation imposed by the aspect-code handler the service returns different results with the same input parameters. Also, the approach cannot be adapted to address user-awareness since the adaptation functionality is hard-coded in the application. Non-satisfied Web Service requirements can be addressed in this approach using one of the proposed techniques (Ortiz et al., 2009). In terms of development automation and service heterogeneity the approach allows generating Web Service skeletons for the main functionality and service-side aspect-oriented code that enables device-specific adaptation. Therefore, the approach reduces significantly the coding effort but does not support the automatic development of client-side GUIs for different platforms. In addition, the approach targets Java and J2ME implementation technologies and does not attest as to the flexibility in adapting code generation for additional platforms.

In contrast to the aforementioned approaches our proposed method provides a fully extensible MDD approach that provides high-degree of automation in developing device-aware Web Services. The proposed approach automates the development of the GUIs and the Web Service proxy classes for different target implementations. Moreover, we have illustrated the extensibility of our approach in terms of adding new graphical features to the PML and also the efficiency provided in defining transformation rules for different implementations. In addition, via the BookStore Web Service prototype we have demonstrated that it is possible through our approach to address heterogeneity. Finally the proposed approach satisfies half of the Web Service requirements, since non-duplicity and user-awareness are not achieved. We also argue that duplicating some code in the Web Service implementation is a small price to pay for achieving adaptation based on the

device. To further clarify this point, advanced-adaptation might be required in cases where it might be essential to show less information on a mobile device; e.g. due to resource limitations. Thus the best option would be to duplicate some methods at the service-side, which would return less amount of information. Concluding, user-awareness is an essential requirement that can be considered in future work.

CONCLUSION

In this work, a Model-Driven framework has been presented that automates the development of device-aware Web Services. The proposed approach allows modelling GUIs using the notation of the Presentation Modelling Language, whereas the key contribution refers to the transformation of PML models to functional code targeting different platforms encountered on mobile and stationary devices. The code generators proposed have been implemented using a set of tools provided by the openArchitectureWare modelling component of the generic MDD environment. Regarding the communication of the client with the Web Service existing code generation tools (e.g. *Axis2 wsdl2java* tool, *.NET wsdl* tool) that support the transformation of WSDL models to corresponding proxy classes have been used.

The developed prototype showcased the applicability and efficiency of the proposed Model-Driven Web Service oriented framework. The efficiency of the approach has been discussed on the basis of the prototype and the results derived using the LoC metric. The proposed Model-Driven Web Service oriented framework consisting of the PML, WSDL and the code generators revealed the capability to address heterogeneity. In particular, the approach enables developers to automatically generate the required source code of Web Service client applications that allow invoking services from different platforms. An interesting extension of this work is to consider the preferences of the user when adapting the Web Service. For instance, a user might want to receive full details of a book even while using a resource-constrained device. Another user might be satisfied simply by receiving in the response message the book's title and price.

REFERENCES

W3C. (2001). *Web services description language (WSDL) specification* v1.1.

Achilleos, A., Georgalas, N., & Yang, K. (2007). An open source domain-specific tools framework to support model driven development of OSS. In *ECMDA-FA, Lecture Notes in Computer Science, Vol. 4530* (pp. 1 – 16).

Achilleos, A., Yang, K., & Georgalas, N. (2008). A model-driven approach to generate service creation environments. In *Proceedings of the IEEE Globecom, Global Telecommunications Conference* (pp. 1 – 6).

Achilleos, A., Yang, K., & Georgalas, N. (2010). Context modelling and a context-aware framework for pervasive service creation: A model-driven approach. *Elsevier Journal on Pervasive and Mobile Computing, Context Modelling. Reasoning and Management*, *6*(2), 281–296.

Balagtas-Fernandez, F. T., & Hussmann, H. (2008). Model-driven development of mobile applications. In *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering*, (pp. 509-512).

Bartolomeo, G., Blefari-Melazzi, N., Cortese, G., Friday, A., Prezerakos, G., Walker, R., & Salsano, S. (2006). 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*, (p. 209).

Dern, D. (2010). Cross-platform smartphone apps still difficult. *IEEE Spectrum*, 2010.

Dunkel, J., & Bruns, R. (2007). Model-driven architecture for mobile applications. In *Proceedings of the 10th international conference on Business information systems*, (pp. 464-477).

Eclipse Foundation Incorporation. (2011). *Eclipse modelling framework*. EMF.

Evermann, J., & Wand, Y. (2005). Toward formalizing domain modelling semantics in language syntax. *IEEE Transactions on Software Engineering*, *31*(1), 21–37.

Gronmo, R., Skogan, D., Solheim, I., & Oldevik, J. (2004). Model-driven Web services development. In *IEEE International Conference on e-Technology, e-Commerce and e-Service*, (pp. 42-45). IEEE Press.

Heines, J. M., & Schedlbauer, M. J. (2007). Teaching object-oriented concepts through GUI programming. In *Proceedings of the 11th Workshop on Pedagogies and Tools - Teaching and Learning Object Oriented Concepts*.

Jelinek, J., & Slavik, P. (2004). GUI generation from annotated source code. *In Proceedings of the 3rd Annual Conference on Task Models and Diagrams*, (pp. 129-136).

Kapitsaki, G. M., Kateros, D. A., Prezerakos, G. N., & Venieris, I. S. (2009). Model-driven development of composite context-aware Web applications. *Elsevier Journal Information and Software Technology*, *51*(8), 1244–1260.

Kapitsaki, G. M., Kateros, D. A., Prezerakos, G. N., & Venieris, I. S. (2009). Model-driven development of composite context-aware web applications. *Information and Software Technology*, *51*(8), 1244–1260.

Kapitsaki, G. M., Kateros, D. A., & Venieris, I. S. (2008). Architecture for provision of context-aware web applications based on Web services. In *IEEE 19th International Symposium on Personal, Indoor and Mobile Radio Communications*, (pp. 1-5).

Kleppe, A. G., Warmer, J., & Bast, W. (2003). *MDA explained: The model driven architecture: Practice and promise*. Boston, MA: Addison-Wesley Longman Publishing Co.

Link, S., Schuster, T., Hoyer, P., & Abeck, S. (2008). Focusing graphical user interfaces in model-driven software development. In *Proceedings of the First International Conference on Advances in Computer-Human Interaction,* (pp. 3-8). IEEE Computer Society.

Object Management Group. (2006). *Object constraint language (OCL) specification* v.2.0.

Object Management Group. (2007). *Unified modelling language (UML) specification* v.2.1.2.

Ortiz, G., & Prado, A. G. (2009). Adapting Web services for multiple devices: A model-driven, aspect-oriented approach. In *Proceedings of the IEEE Congress on Services*, (pp. 754-761).

Ortiz, G., & Prado, A. G. (2009). Mobile-aware Web services. In *International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, (pp. 65-70).

Sauer, S., Drksen, M., Gebel, A., & Hannwacker, D. (2006). *GUIbuilder: A tool for model-driven development of multimedia user interfaces.*

Serral, E., Valderas, P., & Pelechano, V. (2010). Towards the model-driven development of context-aware pervasive systems. *Elsevier Journal on Pervasive and Mobile Computing, Context Modelling. Reasoning and Management*, *6*(2), 254–280.

Sheng, Q. Z., & Benatallah, B. (2005). ContextUML: A UML-based modeling language for model-driven development of context-aware web services. In *International Conference on Mobile Business*, (pp. 206-212). IEEE Computer Society Press.

Singh, Y., & Sood, M. (2009). Model driven architecture: A perspective. In *IEEE International Advance Computing Conference,* (pp. 1644-1652).

KEY WORDS AND DEFINITIONS

Code Generation: Defines the process that enables the transformation of a model to the corresponding implementation code, which can be readily executed on a specific platform.

Metamodelling: The process that guides the definition of a metamodel, which describes the elements, properties and relationships of a particular modelling domain; i.e. domain specific language.

Mobile Services: Define software services that can be accessed and used through mobile or wireless networks from any type of device; smartphone, laptop, etc.

Model-Driven Development: A software development methodology that focuses on the design and implementation of software applications at an abstract platform-independent level.

Service Development: Defines the systematic procedure that includes the phases of requirement analysis, design, implementation and deployment of a software service.

Services Adaptation: Refers to the capability of the software service to be accessible and adapt its behaviour in accordance to the type of mobile client from which it is executed and the context information.

Web Services: Software systems designed to support interoperable computer interaction over a network. They are implemented as application programming interfaces (API) or Web APIs accessed in a standardized way using the XML, SOAP, WSDL and UDDI open standards over an Internet protocol backbone.