A Model-Driven Framework for Developing Web Service Oriented Applications

Achilleas Achilleos, Georgia M. Kapitsaki, and George A. Papadopoulos

Department of Computer Science, University of Cyprus,
75 Kallipoleos Str., Nicosia, CYPRUS
{achilleas,gkapi,george}@cs.ucy.ac.cy
http://www.cs.ucy.ac.cy

Abstract. The advancements made in terms of the capabilities of mobile devices have shifted the interest of service engineering towards frameworks that are able to deliver applications rapidly and efficiently. The development of services that can be fully functional in mobile environments and operable on a variety of devices is an important and complex task for the research community. In this work, we propose a Model-Driven Web Service oriented framework that combines Model-Driven Engineering (MDE) with Web Services to automate the development of platform-specific web-based applications. The importance of this work is revealed through a case study that involves modelling and generation of a representative Web Service oriented mobile application.

Keywords: model-driven, web applications, code generation, mobile services, web services.

1 Introduction

Web Services (WSs) as the most representative implementation of the Service Oriented Architecture (SOA) are usually exploited in the field of Web-based applications for stationary devices. Lately, the advance of the field of mobile computing has introduced the need for developing Web Services that can be consumed through platform-specific clients in different environments, not only static ones, but also those dominating the mobile computing world [1]. End-users, constantly *on the move*, wish to be offered the same choices when working on their smartphones as on desktop devices [2]. In such environments, where users exploit a variety of devices in terms of complexity, size, computational capabilities etc., the need of developing services and applications that can target different mobile platforms arises.

At the same time the wide adoption of Model-Driven Engineering (MDE) from the research community has led to the advance of platforms and tools that facilitate the transformation of models between different abstraction levels resulting to functional code fragments at the final stage. Many of the approaches follow the paradigm of OMG's Model Driven Architecture (MDA) [3]. In MDA, a major separation mentioned includes the Platform Independent Model (PIM)

A. Harth and N. Koch (Eds.): ICWE 2011 Workshops, LNCS 7059, pp. 181–195, 2011.

and the Platform Specific Model (PSM). PIM is a rather abstract representation of a system and contains no information on implementation details. Conversely, PSM represents the system and takes into account platform specific properties. In the research community many attempts evolve around the issue of model transformation, where model transformation can be performed from PIM to PIM, PIM to PSM, PIM to code, etc. Although in MDE the *model* is the basic software component, models that do not result in functional applications have no practical use to the end-users and, consequently, nor to the service providers. For this reason, this paper focuses on the practicalities of the transformation conducted: the model to code step.

Taking the above challenges into account, in the framework of the current work, the issue of mobile application development for different platforms focusing on the Graphical User Interface (GUI) aspect is addressed. A number of code generators targeting different mobile platforms are defined (e.g., J2ME, C#) using as input the application model represented in the Presentation Modelling Language (PML). The PML, which was conceived and presented in previous work of the authors [18], provides means for designing GUI models that facilitate the generation of Web Service clients. The application modelling is complemented by the Web Services Description Language (WSDL) specification. The set of developed generators covers all major mobile and stationary device technologies. Using this approach, the development of functional Web Service Oriented applications for the following categories of devices is supported: (i) Resource-rich devices; e.g., desktops, laptops, (ii) Resource-competent devices; e.g., Netbooks, IPad, Kindle and (iii) Resource-constrained devices; e.g., mobile smartphones such as Google Nexus One, IPhone, HTC Desire, Nokia N8. This category set is based on the categorisation performed by Ortiz et al. [4], which was extended by adding the second category. The importance of the proposed framework lies in the high-degree of automation achieved, which improves the efficiency of the development process, since it allows developers to generate code for various platforms with limited effort.

The rest of the paper is structured as follows: Section 2 gives an overview of the related work in the field, whereas the description of the framework is provided in Section 3. This section is dedicated specifically to the details of the code generation process, which is the main and driving component of the proposed framework. The framework's applicability is exemplified in Section 4 through a mobile bookstore use case. Section 5 concludes the paper.

2 Related Work

In the literature various approaches exploiting the principles of MDE exist, either for the development of software applications in general or specifically for GUI development. Concerning modelling, it is important to keep the application's presentation independent from other layers (i.e., application's logic) so as to facilitate the integration with different technologies. Additionally, modelling GUIs in an abstract way facilitates mapping to different implementations.

Thus, we argue that a framework should provide components (e.g., architecture, code generators) to model and implement independently the GUIs and the WS implementation logic.

Initial work on GUI modelling focuses on the definition of the GUI structure using presentation diagrams and its behaviour using hierarchical statechart diagrams [5]. The definition of GUI structural and behaviour models is supported by the GuiBuilder tool, which allows the transformation from models to Java code. This work focuses simply on the development of Java-based GUIs, which can be used for implementing fully-functional multimedia desktop applications. Other examples of GUI modelling can be found in [6] and [7], although no details on the support for transformations to code are given.

Dunkel and Bruns [8] propose a simple and flexible approach for the development of mobile applications. They present a model-driven approach that allows defining the client's GUIs and the service workflow using graphical models, which are then transformed into XML-based descriptions (i.e., XForms code). The XForms W3C standard has been chosen because of its close correlation with the Mobile Information Device Profile (MIDP) of J2ME, which facilitates the mapping of XForm elements to MIDP elements. The approach is thus tailored towards J2ME and does not exploit the interoperability benefits of Web Services technology. Additional approaches [9], [10] overcome the issues faced by pure XML-based approaches, such as being data centric and the inability to expressively model behaviour, by developing graphical modelling environments that "speak" XML. This imposes though the overhead of developing and maintaining the modelling environments, which is a laborious and costly task. An extended discussion on various solutions for user interface design in application development is present in the survey of Perez-Media et al. [12].

Kapitsaki et al. [11] present an approach that automates the development of composite context-aware Web applications. The defined model-based approach proposes complete separation of the Web application functionality from the context adaptation. In particular, the methodology adopted utilises the Unified Modelling Language (UML) for the design and automatic generation of a functional context-aware Web application. The approach tackles and automates the development of context-aware Web applications, intended mainly for mobile users, which are formulated by third-party WSs. The use of UML should be replaced by standards that provide methods of accessing model stereotypes across different modelling tools.

The heterogeneity of mobile platforms in conjunction with the use of WSs is discussed also by Ortiz et al. [13]. The authors propose a service-side, aspect-oriented approach that allows developers to extend the implemented WS in order to enable the adaptation of the WS invocation result in accordance to the client device. The actual WS code is not directly affected, since additional aspect code is implemented, which intercepts the invocation of the service operation and adapts it according to the device type detected. This approach suffers from three main issues: the client-side implementation needs to include code that allows declaring from which device the WS is invoked, response time is slightly

increased since the service-side aspect code requires to process and adapt the result, and implementation of different platform-specific service-clients is not considered.

In the area of Web development, not specific to Web Services, a number of tools have been proposed offering the means to model Web applications. Good examples can be found in UML-based Web Engineering (UWE[1]), where the application is modelled in UML notation containing a presentation model for the GUI properties, and Object-Oriented Hypermedia Design Method (OOHDM) [14], which targets hypermedia Web applications. Significant works, where Web Service-enabled applications are partially supported, can be found in the WebML CASE tool [15], a visual language used to represent the content structure of a Web application, and Hera [16], which focuses on Web Information Systems and hypermedia applications exploiting tools from the semantic Web, i.e., RDF (Resource Description Framework)) and RDFS. Although most of these works are quite mature, they differ from our approach in the motivation and in the target platforms supported, i.e., stationary devices in the above cases and mobile devices, which are the main focus in our case.

3 The proposed Framework

In this section the main steps of the development process are described; PML is briefly introduced, while particular emphasis is devoted to the transformation step. We emphasise on the specifics of the code generation phase in order to reveal the practicality and applicability of the transformation approach, which enables targeting different mobile but also stationary platforms. The description of the earlier steps is required and has been included in order to provide a comprehensive view of the proposed model-driven Web Service oriented framework.

3.1 Scope of Use and Overall Development Process

The proposed development process combines the characteristics of Web Services with the development directives given by Model-Driven Engineering. The presentation layer and the Web Service layer are kept distinct, in order to allow each one to be mapped to different implementations. This transformation logic is presented in Fig. 1, where each client is defined and developed in the form of GUIs and collection of Web Service communication classes. In the presentation particular focus is given on the GUI-related part depicted on the left side of the figure. Note that the Web Service implementation is conducted by the developer through a manual process. Nevertheless, the technology employed for implementing the WS main functionality does not restrict the client implementation to a specific platform. This is because Web Services allow the exchange of XML-based messages between entities regardless of the implementation details or the programming language used for the WS development.

[1] http://uwe.pst.ifi.lmu.de/

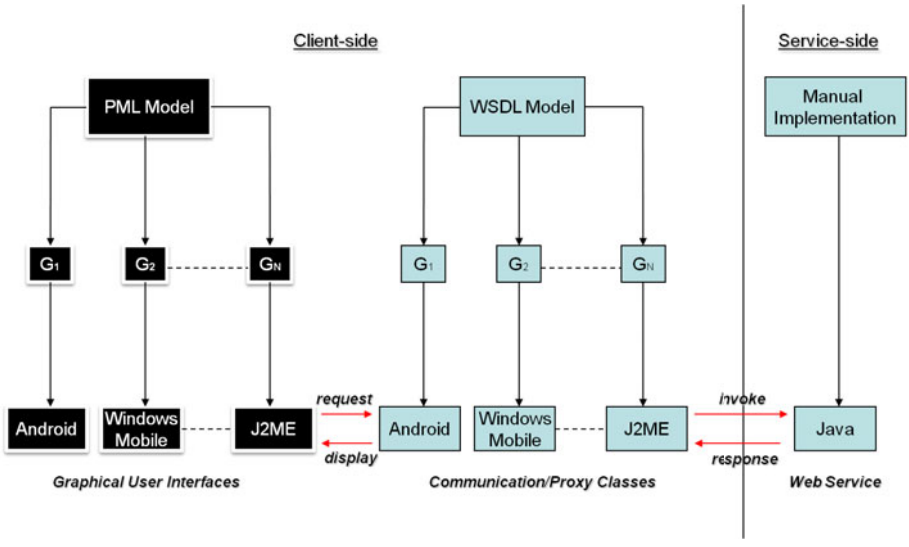

Fig. 1. Model-Driven, Web Service-oriented Architecture

The PML and WSDL models are designed at modeling time and act as input for the code generators that produce the respective code fragments. Regarding the WS part, existing WSDL code generation tools are used that enable the transformation of WSDL models to platform-specific proxy classes exploited for sending and receiving information via WS request and response messages.

In particular, the PML allows modelling GUIs in the form of screen layouts; as desired by the developer. The presentation models include the necessary abstract information on GUI elements (e.g., text box, label), properties (e.g., label's text) and relationships (e.g., panel contains button) of existing major mobile and stationary devices and platforms. A brief overview of PML is provided next. In accordance to the PML notation a number of platform specific code generators have been implemented for the technologies indicated in Fig. 1.

3.2 Brief Overview of PML

The Presentation Modelling Language is defined as an Eclipse Modeling Framework (EMF) based metamodel, using the Graphical Modelling Framework (GMF) Ecore diagram tool included in the environment presented in previous work [17]. The PML metamodel is presented in Fig. 2 and describes the graphical modelling elements, their associations and graphical properties, which enable the design of GUIs in the form of visual abstract models. The metamodel definition is based on an analysis and evaluation performed to identify elements, properties and associations that share similarities across different major platforms. Fig. 3 showcases that the metamodel definition is complemented by the Model-2-Code

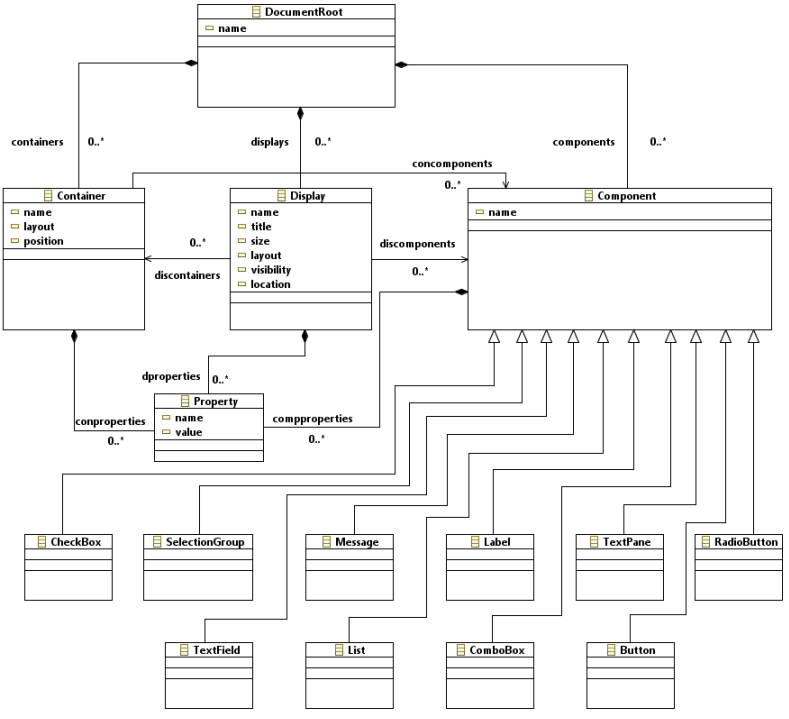

Fig. 2. Presentation Modelling Language metamodel

(M2C) transformation rules, which map abstract PML elements, properties and associations to platform specific implementation components, properties and associations (i.e., platform-specific code).

Most elements defined in the metamodel are self-explainable. *DocumentRoot* is the basic metaclass, where the rest of the model elements are aggregated, such as a number of displays corresponding to the screen of the mobile device (metaclass *Display*). The *discontainers* aggregation defines the containment relationship between the display and its container elements. The common graphical components are defined as children of the *Component* metaclass (e.g., *Message, Label, Button*). Similarly typical associations that exist between objects, such as the fact that container (e.g., panel) *includes* component (e.g., label), are also visible in the metamodel. The *Property* metaclass is an important element of the PML since it allows describing different graphical properties for the modelling elements. Each modelling element may contain various graphical properties, which are defined as instances of the *Property* metaclass. This provides the capability to extend easily and efficiently the PML by introducing new properties simply by adding parsing support within code generators.

The elements included in instances of the PML metamodel are analysed based on a number of transformation rules defined in the code generators. A deeper

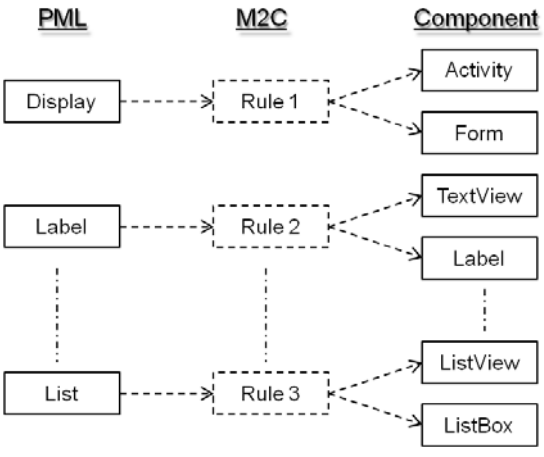

Fig. 3. Mapping PML to platform-specific implementations

analysis on the elements comprising PML is not included in the current paper. More information on the initial version of PML can be found in [18].

3.3 The Transformation Mechanism

The generation of the Web Service invocation part (from the client) is not addressed; instead, existing literature works on generating this part from WSDL descriptions are employed. 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 details. 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 has not been included, since the focus is given on the applicability of the presentation code generation tools on multi-platform environments. However, some works that exploit WS models and introduce tools for model transformation procedures in the framework of MDE exist. The reader can refer to relevant publications [19], [11].

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 the following platforms of stationary and mobile devices have been

implemented: Java, J2ME, Android, Windows Mobile and Windows Desktop. In this subsection two of the above generators have been chosen for demonstration purposes; specifically, the versions targeting Android and Windows Mobile are described in order to showcase also the main differences between the two technologies. In order to keep the paper comprehensive and due to space limitations, it is not possible to describe the whole generators set.

The Eclipse-based MDE environment, proposed in previous work [17], includes the openArchitectureWare (oAW) software tool that enables the development of code generators by defining model-to-text transformation rules. The tool comprises the *Xpand* template language, a text-editor, the workflow execution engine and two supplementary languages (i.e., *Check*, *Xtend*) with their individual text editors. Foremost, the *Xpand* language supports the definition of advanced code generators as templates, which capture the transformation rules and control the output document generation (e.g., XML, Java, C#, HTML). The transformation rules are defined using the *Xpand* text-editor and include references to extension functions specified using the *Xtend* language. In particular, extension functions are considered as utility functions (i.e., similar to Java utility functions), which 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 on the basis of the defined templates and the input model.

The combination of the components supports the code generation process as depicted in Fig. 4. The transformation is executed via the workflow engine of oAW, on the basis of a workflow script that specifies information, such as the classes and components participating in the generation, output folders etc. The template definition, which drives code generation, constitutes the most important part of the transformation process. Appropriate templates have been defined for all participating platforms (i.e., C#, Java, J2ME, Android).

Listing 1 presents a sample part of the Android-specific template definition that allows demonstrating how code generation is achieved. Lines containing information such as generated files and package names have been omitted. 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

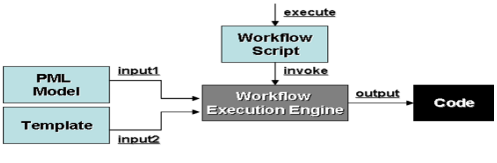

Fig. 4. The PML code generation process

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 the lines that follow (36-72). Depending on the type of element visited during the parsing of the PML model (indicated by the properties of *concomp*), the respective object creation with the appropriate name is generated. For example a new *TextView* object corresponds to each *Label* model element as indicated in lines 37-39, where the *keyword/property "text"* used at line 39 provides the capability to set the text on the label to the value parsed from the *Label* modelling element. The list of conditional statements allows to parse and generate other types of secondary components using the same reasoning.

Listing 1. Sample for the Android-specific template

```
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>>
     .....
98.  <<ENDDEFINE>>
```

For the template definition targeting Windows Mobile a sample part is illustrated in Listing 2. The same approach has been employed for the remaining platform-specific code generators.

Listing 2. Sample for the Windows mobile-specific template

```
1.  <<EXTENSION templates::WindowsMobilePresentation>>
2.  <<DEFINE Root FOR presentation::DocumentRoot>>
    .....
23. <<REM>>Create the constructor that creates Windows mobile main form.<<ENDREM>>
24. public <<this.toFirstUpper()+"WindowsMobile">>(){
25. <<FOREACH discontainers AS discon->>
26. <<REM>>Set the name and title of the Windos mobile main form.<<ENDREM>>
27.       this.Name = <<discon.conproperties.select
                    (e|e.name.contains("title")).value.first()>>;
28.       this.Text = <<discon.conproperties.select
                    (e|e.name.contains("title")).value.first()>>;
29. <<REM>>Create the components associated to each layout of the main Form.<<ENDREM>>
30. <<FOREACH discon.concomponents AS concomp ITERATOR it->>
31. <<IF concomp.metaType.name.matches("presentation::Label")->>
32.       <<concomp.name>> = new Label();
```

```
33.       <<concomp.name>>.Name = "<<concomp.name>>";
34.       <<concomp.name>>.Location = new System.Drawing.Point(0, 0);
35.       <<concomp.name>>.Size = new System.Drawing.Size(0, 0);
36.       <<concomp.name>>.TabIndex = <<it.counter0>> ;
37.       <<concomp.name>>.Text = <<concomp.compproperties.select
                  (e|e.name.contains("text")).value.first()>>;
39. <<ELSEIF concomp.metaType.name.matches("presentation::TextField")->>
40.       <<concomp.name>> = new TextBox();
41.       <<concomp.name>>.Name = "<<concomp.name>>";
42.       <<concomp.name>>.Location = new System.Drawing.Point(0, 0);
43.       <<concomp.name>>.Size = new System.Drawing.Size(0, 0);
44.       <<concomp.name>>.TabIndex = <<it.counter0>>;
45.       <<concomp.name>>.Text = "";
46. <<REM>> Ends the loop associated with the components collection. <<ENDREM>>
47. <<ENDFOREACH>>
48. <<REM>> Ends the loop associated with the containers collection. <<ENDREM>>
49. <<ENDFOREACH>>
    .....
70. <<ENDDEFINE>>
```

4 The Book Store Use Case

4.1 Overview

The chosen use case consists of a *BookStore* WS that provides means for searching and purchasing books. Specifically, the user exploiting the service can search for books, providing as input the book title. The WS returns all details of the book and gives the opportunity to the user to purchase the book. This latter operation is invoked by filling a number of necessary fields to complete the order and payment (including the customer information and the details of the payment method). Upon successful completion of the transaction a result page is shown.

Although not directly exploited in the context of the current work, the model of the *BookStore* WS is depicted in Fig. 5. The server side part of the *BookStore* prototype has been manually implemented in Java. The demonstration of this section concentrates on the generation of the presentation layer of the client side.

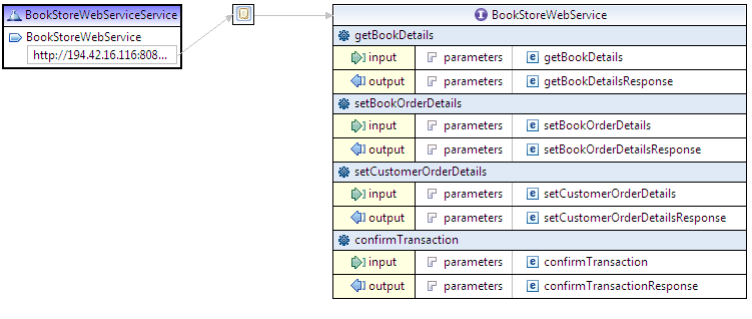

Fig. 5. The *BookStore* Web Service Description Language Model

4.2 Models Design and Code Generation

Due to the complexity of the WS and space limitations the PML model, which is degined manually by the application developer, is not provided in full; its basic parts demonstrating the use of the PML metamodel are given instead. The modeling part of the containers corresponding to distinct screens is shown in Fig. 6. At the top of the model an instance of the *Display* metaclass represents the main frame/display of the GUI. The display is associated with a number of container components that form instances of the *Container* metaclass. The first container, i.e., *searchForBooks*, corresponds to the first step of book searching, whereas the rest serve the book purchasing procedure.

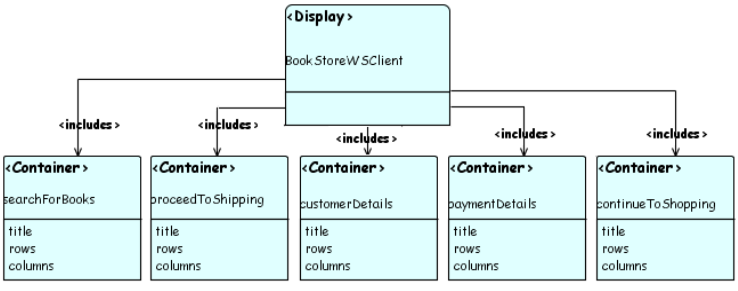

Fig. 6. Top level elements of the *BookStore* Presentation Model

Each container has its own properties and contains also secondary components, i.e., label, textfield, textpane and button elements, as shown for a specific container in Fig. 7. The secondary components are defined as instances of the respective metaclasses and include their own graphical properties. The *customerDetails* container corresponds to the phase, where the customer needs to provide as input her details with information, such as name and shipping address. These fields correspond to different GUI element types and are marked appropriately in the model.

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

```
1.  /** Called when the activity is first created. */
2.  public View searchForBooksView() {
3.        this.setTitle("BookStore - Multi-platform Web Service");
4.        searchForBooks = new TableLayout(this);
5.        bookTitle = new TextView(this);
6.        bookTitle.setText("Enter Book Title:");
7.       titleOfBook = new EditText(this);
8.      ... }
```

A segment of the generated code for the Android platform corresponding to lines 31-74 of Listing 1 is shown in Listing 3. These lines of code are the

outcome of the transformation of the modelling elements corresponding to the *Display* and the individual graphical properties of two of the components of the searchForBooks *Container*. Although hidden in the models presented above, the name provided for the book store (i.e., *BookStoreWSClient*) forms part of the information that can be edited through the model properties editor view. In a similar fashion the generated code fragment corresponding to Windows Mobile is visible in Listing 4 following the specifics of the platform.

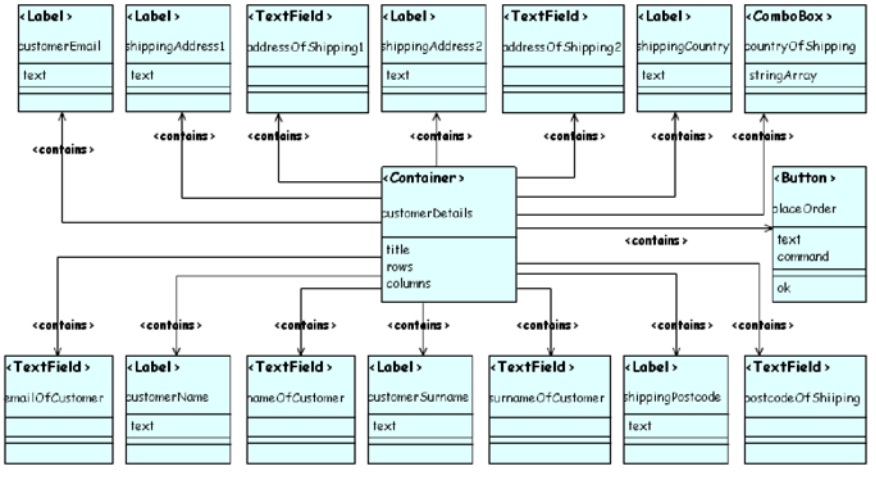

Fig. 7. Example container of the *BookStore* Presentation Model

Fig. 8 demonstrates some of the screenshots captured during the use of the *BookStore* WS on mobile clients deployed on the Android and the Windows Mobile platform. The screens for searching for a book, displaying the results and filling out the information for purchasing the book are shown. Alternated screenshots capture different steps in the functionality of the WS, while running on these platforms. Moreover, Table 1 presents a quantitative evaluation by comparing the generated code against the complete implementation (i.e., including manual code) for the examined platforms. Manual implementation for each platform is limited to: (i) calling the method that displays the next form, (ii) getting user input from form fields, (iii) calling the appropriate WS method (passing user input as arguments) via the proxy class and (iv) obtaining and displaying the WS response. The Lines of Code (LoC) belongs to the size metrics that can be used for analyzing the quality of model transformations in MDE. In future work, we aim to extend the evaluation by examining other software metrics, such as software cyclomatic complexity, and include a performance analysis evaluation that can provide more accurate results than the LoC metric.

Fig. 8. The *BookStore* Web Service Deployed on Different Devices

Table 1. LoC percentage for the different platforms

LoC Metric per Platform	Generated Code	Overall Code	Generated/ Overall (%)
Java	189	334	56.59
J2ME	267	369	72.36
Android	244	361	67.59
Windows Mobile	360	481	74.84
Windows Desktop	360	475	70.30
All Platforms	1420	2020	70.30

Listing 4. The GUI code generated for the Windows mobile target platform.

```
1.  public BookStoreWSClientWindowsMobile(){
2.        this.Name = "BookStore - Multi-platform Web Service";
3.        this.Text = "BookStore - Multi-platform Web Service";
4.        bookTitle = new Label();
5.        bookTitle.Name = "bookTitle";
6.        bookTitle.Location = new System.Drawing.Point(20, 20);
7.        bookTitle.Size = new System.Drawing.Size(200, 20);
8.        bookTitle.TabIndex = 0;
9.        bookTitle.Text = "Enter Book Title:";
10.      titleOfBook = new TextBox();
11.      titleOfBook.Name = "titleOfBook";
12.      titleOfBook.Location = new System.Drawing.Point(20, 45);
13.      titleOfBook.Size = new System.Drawing.Size(200, 20);
14.      titleOfBook.TabIndex = 1;
15.      titleOfBook.Text = "";
16.      ...    }
```

The results show that each generator can be exploited in an efficient way, in order to automate a significant part of the implementation process (the average percentage is kept at 70.3%). Note that in the case of Java the percentage is lower, since it is not possible to generate the code that handles components

placement for each screen. This is due to the layout choices offered by Java, which are missing in other technologies. For this reason the respective lines of code in Java need to be added manually.

5 Conclusions

In this work, a Model-Driven framework that automates the development of Web Service oriented applications has been presented. The process described allows modelling service-client GUI elements using the notation of the Presentation Modelling Language, whereas the key contribution refers to the transformation of PML models to functional code, targeting the different platforms encountered on mobile and stationary terminals (Java, Android, etc.). The code generators proposed have been implemented using a set of tools offered by the openArchitectureWare modelling component. Regarding the communication of the client with the Web Service, existing tools that support the transformation of WSDL models to corresponding proxy classes have been used.

The developed prototype and its applicability have been demonstrated through the book store WS, which was showcased running on mobile environments but deployed also on desktop devices (i.e., Java, Windows Desktop). The efficiency of the approach has been discussed on the basis of the use case and the results derived using the LoC metric. The proposed model-driven WS-oriented framework, consisting of the PML and WSDL along with the code generators set, has the capability to address heterogeneity when developing platform-specific applications. In particular, the approach allows users to automatically generate the required source code for Web Service client applications, which can consequently invoke Web Services from different devices and platforms. An interesting extension of this work is to consider the preferences of the user when adapting the Web Service, making this way the service user-aware. For instance, a user might want to receive the full book details even if she is using a resource-constrained device, while another user is satisfied with receiving the book's title and price.

References

1. Bartolomeo, G., Blefari Melazzi, N., Cortese, G., Friday, A., Prezerakos, G., Walker, R., Salsano, S.: SMS: Simplifying Mobile Services - for Users and Service Providers. In: Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services, p. 209. IEEE Computer Society, Washington (2006)
2. Dern, D.: Cross-Platform Smartphone Apps Still Difficult. In: IEEE Spectrum. IEEE Press (2010)
3. Singh, Y., Sood, M.: Model Driven Architecture: A Perspective. In: IEEE International Advance Computing Conference, pp. 6–7. IEEE Computer Society (2009)
4. Ortiz, G., Garcia de Prado, A.: Adapting Web Services for Multiple Devices: A Model-Driven, Aspect-Oriented Approach. In: IEEE Congress on Services, pp. 754–761. IEEE Computer Society, Los Alamitos (2009)

5. Sauer, S., Duerksen, M., Gebel, A., Hannwacker, D.: GuiBuilder: A Tool for Model-Driven Development of Multimedia User Interfaces. In: Workshop on Model Driven Design of Advanced User Interfaces in MODELS 2006 (2006)
6. Link, S., Schuster, T., Hoyer, P., Abeck, S.: Focusing Graphical User Interfaces in Model-Driven Software Development. In: First International Conference on Advances in Computer-Human Interaction, pp. 3–8. IEEE Computer Society, Washington (2008)
7. da Cruz, A.M.R., Faria, J.P.: A Metamodel-Based Approach for Automatic User Interface Generation. In: Petriu, D.C., Rouquette, N., Haugen, Ø. (eds.) MODELS 2010. LNCS, vol. 6394, pp. 256–270. Springer, Heidelberg (2010)
8. Dunkel, J., Bruns, R.: Model-Driven Architecture for Mobile Applications. In: Abramowicz, W. (ed.) BIS 2007. LNCS, vol. 4439, pp. 464–477. Springer, Heidelberg (2007)
9. Paternó, F., Santoro, C., Spano, L.D.: User task-based development of multi-device service-oriented applications. In: International Conference on Advanced Visual Interfaces. LNCS, vol. 5726. ACM (2010)
10. Paternò, F., Santoro, C., Spano, L.D.: Model-Based Design of Multi-Device Interactive Applications Based on Web Services. In: Gross, T., Gulliksen, J., Kotzé, P., Oestreicher, L., Palanque, P., Prates, R.O., Winckler, M. (eds.) INTERACT 2009. LNCS, vol. 5726, pp. 892–905. Springer, Heidelberg (2009)
11. Kapitsaki, G.M., Kateros, D.A., Prezerakos, G.N., Venieris, I.S.: Model-driven development of composite context-aware web applications. Information and Software Technology 51(8), 1244–1260 (2009)
12. Pérez-Medina, J.-L., Dupuy-Chessa, S., Front, A.: A Survey of Model Driven Engineering Tools for User Interface Design. In: Winckler, M., Johnson, H. (eds.) TAMODIA 2007. LNCS, vol. 4849, pp. 84–97. Springer, Heidelberg (2007)
13. Ortiz, G., Garcia de Prado, A.: Mobile-Aware Web Services. In: International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, pp. 65–70. IEEE Computer Society, Los Alamitos (2009)
14. Moura, S.S., Schwabe, D.: Interface Development for Hypermedia Applications in the Semantic Web. In: LA Web, pp. 106–113. IEEE CS Press (2004)
15. Brambilla, M., Ceri, S., Comai, S., Fraternali, P.: A CASE tool for modelling and automatically generating web service-enabled applications. International Journal of Web Engineering and Technology 2(4), 354–372 (2006)
16. van der Sluijs, K., Houben, G.J., Leonardi, E., Hidders, J.: Hera: Engineering Web Applications Using Semantic Web-based Models. In: de Virgilio, R., Giunchiglia, F., Tanca, L. (eds.) Semantic Web Information Management - A Model-Based Perspective, pp. 521–544. Springer, Heidelberg (2010)
17. Achilleos, A., Yang, K., Georgalas, N.: A Model Driven Approach to Generate Service Creation Environments. In: IEEE Global Telecommunications Conference, pp. 1–6. IEEE (2008)
18. Achilleos, A.: Model-driven Petri Net based Framework for Pervasive Service Creation. School of Computer Science and Electronic Engineering. University of Essex (2010)
19. Gronmo, R., Skogan, D., Solheim, I., Oldevik, J.: Model-driven Web services development. In: IEEE International Conference on e-Technology, e-Commerce and e-Service, p. 42045. IEEE Press (2004)
20. van Amstel, M.F., Lange, C.F.J., van den Brand, M.G.J.: Metrics for Analyzing the Quality of Model Transformations. In: 12th ECOOP Workshop on Quantitative Approaches on Object Oriented Software Engineering (2008)