A Model Driven Approach to Generate Service Creation Environments

A. Achilleos[1], K. Yang[1], N. Georgalas[2]
[1]University of Essex, Dept. of Computing and Electronic Systems, UK
{aachila,kunyang}@essex.ac.uk
[2]British Telecom Group, UK
nektarios.georgalas@bt.com

***Abstract*— The creation of services is a complex activity that involves several tasks. Furthermore this complexity is augmented by the fact that supporting service creation environments are technology-specific. Consequently a technology-independent approach and framework are required to generate service creation environments and drive service creation. In this paper we present such an approach and a generic framework for supporting service creation. The approach realizes service creation via the phases of: (i) domain specific language definition, (ii) model definition and validation, (iii) model-to-model transformation and (iv) model-to-code generation. Each phase maps to a corresponding phase in service creation starting from service analysis to service implementation. The applicability of the approach and its accompanying framework is demonstrated via an example scenario that illustrates the automatic generation of a service creation environment for an online survey system.**

***Index Terms*— Service creation environment, domain specific modelling, model driven development.**

I. INTRODUCTION

Service creation refers to the set of activities involved in the rapid analysis, design, implementation, and validation/testing of services [1]. Typically it denotes a service creation process that guides in a systematic way the rapid creation of services. This process is normally bound to a service creation environment in order to be effectively applied in practice. Many frameworks have been built in an attempt to simplify service creation. These frameworks are considered complex in nature since they are tailored around a specific technology. Explicit knowledge of the technology involved is required something that non-expert users may not have [2]. Such types of frameworks therefore benefit experts that are familiar with a particular technology (e.g. a programming language) but undoubtedly do not aid non-expert users.

Subsequently high-level frameworks are required that can assist both expert and non-expert users. These types of frameworks can offer a higher level of abstraction since they are not bound to any particular implementation technology. Therefore they are not hindered by technological complexities and thus can simplify and support effectively the service creation process. Domain Specific Modelling (DSM) [3], [4] and Model Driven Architecture (MDA) [5], [6], [7] are the two modelling paradigms combined in the context of this paper for creating such a generic framework [8].

DSM refers to a software engineering methodology that raises the level of abstraction by introducing models as the prime entities of the software development process [9]. These models are defined using a Domain Specific Language (DSL) [10], [11] which includes the artefacts and the notation applicable to a particular problem domain. DSM objective is to automatically generate the implementation from these models. The problem with DSM though is the difficulty faced with the design, development and application of DSLs. Developing a DSL and implementing its DSM environment from scratch is often difficult and costly.

MDA depicts a similar software engineering approach in which software systems are expressed as UML platform independent models (PIMs). These PIMs are then transformed to platform specific models (PSMs) that include technology specific details. Ultimately from these PSMs the platform specific implementation is generated. MDA considers furthermore the necessity to automate the process of developing modelling software tools to benefit substantially from any modelling approach. On the other hand focusing on a particular application domain, as in the case of DSM, rather than the generic UML language increases the effectiveness and efficiency of the approach.

Consequently, MDA with DSM integration allows benefiting from each approach and defining a coherent Model-Driven Development (MDD) methodology to support service creation. In overall the methodology supports the following stages: DSL definition, model definition and validation, model-to-model transformation and model-to-code generation. Each step of the MDD process maps accordingly and accomplishes a particular phase of service creation.

The main contributions of this paper lie in a twofold. Firstly, an MDD methodology is defined, for the first time to the best of our knowledge, to support the service creation process. Secondly, a generic framework built on top of the Eclipse platform is presented, which facilitates the generation of service creation environments. In this context the phases of the MDD process bound to the framework are presented revealing the mapping to the service creation phases. Furthermore we illustrate the applicability of the framework in effectively accomplishing those tasks via a realistic case study for creating an online survey system.

The remainder of this paper is structured as follows: Section 2 presents related work on service creation and service creation environments. In Section 3 we introduce the model-driven methodology for supporting service creation and Section 4 introduces the framework features and capabilities. Section 5 presents a case study that illustrates the generation of a service creation environment for an online survey system. Finally, we present the conclusions and future work.

978-1-4244-2324-8/08/$25.00 © 2008 IEEE.

II. RELATED WORK

Research work on service creation has revealed that a service creation methodology and a high-level service creation framework are required to enable rapid development of advanced services.

One of the initial efforts was made by Bernhard Steffen et al [12], which designed and implemented a constraint-oriented service creation environment for the creation of advanced telephony services. The environment is based on MetaFrame, an environment designed for the flexible management of large repositories of complex components. In the proposed approach a developer with advanced programming skills is required to develop individual components (libraries) using the environment. A designer loads the libraries and using a graphical representation composes the service from the individual developed components. The service is then validated against a variety of constraints to ensure its correctness and guarantee its executability.

Analogous approaches [13], [14] have been proposed that rely also on technology-specific frameworks to aid and simplify the rapid creation of value-added services. Although these approaches reduce time and cost to create new services, they target experienced developers of the particular technology involved and they certainly do not assist non-expert users.

In an attempt that differentiates from the above, John-Luc Bakker and Ravi Jain [15] propose an XML-based platform-independent approach. They present a standard scripting language, the Service Creation Markup Language (SCML), which integrates a family of languages to model essential service aspects (e.g. user interaction, mobility). The SCML is supported within the service creation environment of the JAIN industry forum. Despite its abstract approach, the languages and the software tools of the environment require to be manually developed, something that introduces a time consuming and costly process. Furthermore the XML representation of the languages can be devious to comprehend in order to realise effectively different service aspects.

Roch H. Glitho et al [2] in their work present an approach and a generic framework, which also steers clear of technological complexities. The framework facilitates service definition in an abstract manner using a set of predefined Java components and a GUI with drag and drop capabilities. From this service definition the service implementation is automatically generated in the form of Java classes. In this work though the framework is once again implemented manually and subsequently any future extension (e.g. with additional Java components) requires to manually re-implement the framework.

Dionisis X. Adamopoulos et al [1] examine existing issues in aiding and simplifying the creation of telematic services. The authors identify that a systematic service creation methodology and a service creation environment are necessary to facilitate the efficient and rapid creation of services. In their approach they propose a methodology and an environment composed of different software tools to support the process. The methodology, although well-formed, it relies on multiple sub-processes something that hinders the approach increasingly complex. Furthermore, the supporting service creation framework relies mainly on the UML standard for performing the service analysis and design phases. Therefore the modelling approach still remains at a generic level and is not effectively confined to a particular application domain.

According to our views a coherent methodology should allow to carry out effectively the service creation phases at a platform independent level. Additionally it should provide the capability to generate automatically a service creation environment, which can support the service creation process.

The proposed framework is composed by integrating individual open-source modelling projects on top of the Eclipse platform. The selection of the Eclipse platform was made due to its widely accepted component-based architecture, which serves as the foundation for building an easily extensible meta-framework. Various existing major meta-modelling frameworks such as Borland Together, Generic Modelling Environment (GME) and XMF-Mosaic are also built on top of the Eclipse platform. Another major framework is Microsoft DSL tools, which is build on top of the .NET platform. It is itself a powerful commercial product that relies though on its own meta-modelling libraries and does not follow to the Object Management Group (OMG) MDA standards; as in the case of the framework proposed in this paper.

III. MODEL DRIVEN DEVELOPMENT METHODOLOGY

The model-driven development process presented in this paper aims to provide a systematic methodology that facilitates the generation of service creation environments and supports in overall the service creation process. Table 1 reveals the mapping between the MDD process phases and the service creation phases.

TABLE 1. MAPPING THE MDD PROCESS TO SERVICE CREATION.

MDD process	***Service creation process***
Domain specific language definition	Service analysis
Domain model definition	Service design
Domain model validation	Service validation/testing
Domain model-to-model transformation	Service implementation/ management
Domain model-to-code generation	Service implementation

A. Domain Specific Language Definition

Primarily the DSL definition phase reflects the service analysis phase. In this phase the service is decomposed into its elements and their relationships, in an attempt to provide an overall understanding of the service. In the same exact manner a DSL is defined in the form of a domain meta-model using a meta-meta-language. The meta-model is the description of the modelling language, which is defined by identifying the semantics of the particular application domain and mapping them to language constructs. Furthermore domain rules that govern the language are identified, mapped to constraints and imposed onto the language constructs. The constraints are most commonly applied explicitly and separately of the domain meta-model definition; some rules are implicitly stated in the definition.

Correct definition of the modelling language and successful imposition of constraints allows defining complete and unambiguous models. Since subsequent phases of the development process rely heavily on these models, it is crucial that the language allows the definition of coherent models. This saves considerably on time, effort and minimises costs, since it does not require performing heavy testing and applying extensive corrections onto the generated implementation.

Domain models definition requires a graphical modelling environment, which is part of the overall service creation environment. Implementing a modelling environment from scratch is often cumbersome and costly. Especially its maintenance introduces quite an overhead on the development process and subsequently increases costs. Therefore the generic framework must provide the capability to generate automatically the modelling environment.

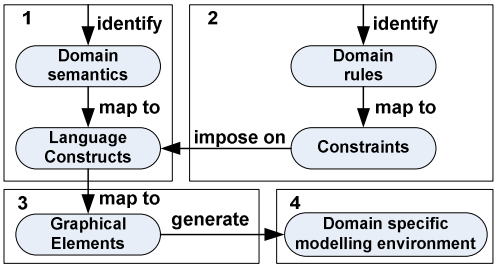

Fig. 1. Domain specific language definition.

This is performed by mapping the semantics of the domain meta-model into specific graphical elements, forming the graphical meta-model. Both the domain and graphical meta-models define the entire set of artefacts required to enable the automatic generation of the modelling environment. The environment comprises an editor, bound to the semantics of the language, and provides validation capabilities for the stated rules. Figure 1 demonstrates the four initial activities undertaken to effectively accomplish the language definition and generate the modelling environment (1-4).

B. Domain Model Definition and Validation

During the second phase the generated environment facilitates domain models definition via the use of the modelling editor. This phase corresponds to the service design phase since models can resemble services; if the DSL(s) targets the application domain of services.

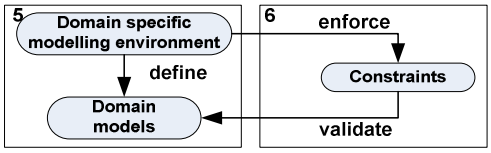

Fig. 2. Domain model definition and validation.

The next phase involves domain models validation against the rules set during the language definition phase. With the use of the generated environment constraints are effectively enforced. As a result only non erroneous implementations can be automatically generated from the validated models. Models validation reflects the service validation/testing phase if again we consider that models resemble services. Figure 2 illustrates the model definition and validation phases (5-6).

C. Model to Model Transformation

The process involves model-to-model transformations, which assist either the service implementation or the overall service lifecycle management. During the implementation phase the source model can be transformed to a target platform specific model (PSM). The PSM includes technology specific details and conforms to a platform-specific language, which reflects the semantics of a programming language. With the use of a transformation language the mapping between the semantics of the source and target language is defined to enable execution of the transformation.

In case the intermediary transformation phase is omitted implementation specific details are hard-coded inside the code generator; see case study. Besides the service implementation phase, transformations can also be used for the configuration management of services when porting from one service version to another version. Model-to-model transformation phase (7), presented in Figure 3, is not considered in this paper, although it is fully supported by the framework.

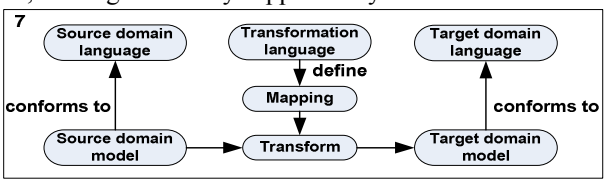

Fig. 3. Model-to-model transformation.

D. Model to Code Generation

The final phase is model-to-code generation, which corresponds to the service implementation phase. The implementation is obtained via an automatic or semi-automatic process that transforms the models to service code. For the interpretation of the model to executable code a platform-specific code generator needs to be defined in the form of a template. The generator is responsible for interpreting the semantics of the domain model into the corresponding software semantics of a programming language. Figure 4 illustrates the model-to-code generation phase (8).

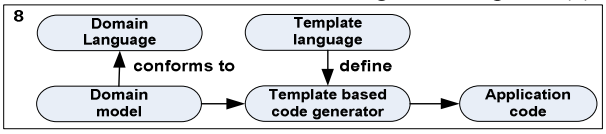

Fig. 4. Model-to-code generation.

IV. THE PROPOSED GENERIC FRAMEWORK

To apply the methodology in practice a supporting generic meta-modelling framework is required. The proposed framework has an open source and extensible component-based architecture. It is basically a suite of software components integrated together on top of the Eclipse platform. Each component aids a particular phase of the process via the software tools it provides. These components are namely the *Eclipse Modelling Framework (EMF)* [16], the *Graphical Modelling Framework (GMF)* [17], the *Atlas Transformation Language (ATL)* [18] and *openArchitectureWare (oAW)* [19]. Figure 5 illustrates the components and their key software tools described next, revealing how they accomplish each step of the process.

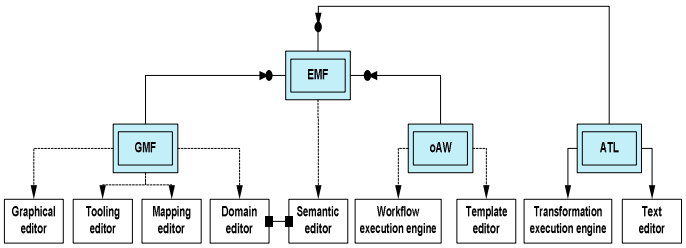

Fig. 5. Framework architecture.

EMF and GMF are the meta-modelling components that support the DSL definition phase. At the heart of the language definition is the EMF meta-meta-language, which is regarded as an implementation of the Meta Object Facility (MOF) [20]. It is considered itself as a DSL that facilitates the definition of DSLs. EMF provides a tree-based semantic editor tool that practically allows the definition of the new language. The tree-based representation of the meta-model makes it very difficult though to comprehend the DSL defined.

Therefore the GMF is used instead for the meta-model definition because it provides a visual extension of the EMF semantics. Its domain editor aids DSL definition using graphical elements providing a visual representation of the semantic meta-model. It is denoted that the semantic meta-model is automatically created and updated with respect to any modification performed in the visual domain meta-model.

Complementary to the semantics definition is the imposition of complex rules that govern the meta-model. Rules are applied onto the semantic meta-model definition using the Object Constraint Language (OCL) [21] feature of the mapping editor. The rules imposed in the form of constraints restrict the modelling language and permit only valid models to be defined using the language. Constraints turn the informal meta-model definition into a formal modelling language.

Furthermore the GMF facilitates automatic extraction of both a graphical and a tooling meta-model, which can be manipulated using the graphical and tooling editors. The former depicts the drawing figures of the editor of the DSM environment and their mapping to the semantic constructs of the DSL. The latter on the other hand associates elements and relationships of the domain language with corresponding palette components that enable the drag-and-drop functionality of the editor. From the mapping of the three meta-models the DSM environment is automatically generated.

The environment is generated in the form of an Eclipse plug-in, something that allows importing it as a new feature of the generic framework. Hence domain model definition is enabled using the modelling editor of the environment. Furthermore the environment provides the capability to enforce the constraints defined and validate the integrity of the models. In case a specific inconsistency is detected a suitable error message is displayed onto the editor suggesting possible resolutions using an informative description.

Transformations are defined and executed using another component of the framework, the Atlas Transformation Language. The ATL component comprises of a DSL for writing transformations and the transformations execution engine. Model-to-model transformations are defined using the provided text editor tool on the basis of the ATL language. The transformation is defined in the form of a mapping between the semantics of the meta-models involved in the transformation. Following the ATL execution engine accepts as inputs the mapping and the input DSL model, and executes the transformation to the corresponding output DSL model.

Concluding we have the model-to-code generation phase, which is delivered via the oAW component. It comprises the xPand language, an editor tool and the workflow execution engine. xPand is a template language, which supports advanced features for building code generators for any programming language. Code generators are built on the basis of the meta-model definition in the form of templates, defined using the template editor. The workflow engine is responsible to execute the transformation of the model to a particular implementation. It accepts as inputs the template-generator and the model and transforms the model to the designated programming language.

V. CASE STUDY: CREATION OF AN ONLINE SURVEY SYSTEM

The proposed framework guides the generation of service creation environments and supports service creation. This section attests to this claim by illustrating the applicability of the framework by generating a service creation environment for the development of an online survey system. It must be denoted that this exemplary case study aims merely at evaluating the methodology and the capabilities of the framework. To effectively define a service it is required to generate an environment with different modelling languages that facilitate the definition of distinct service aspects; such as data, behaviour and user interaction modelling.

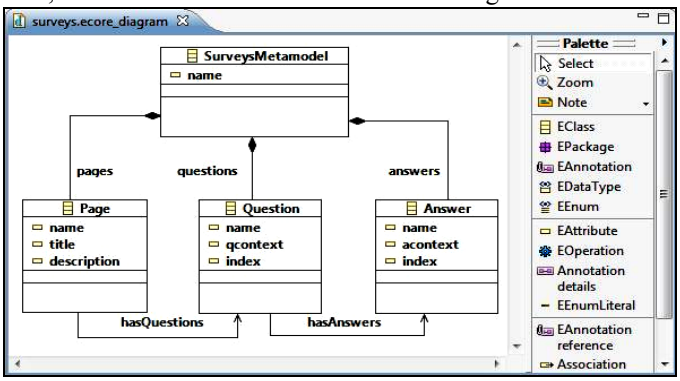

Fig. 6. Survey meta-model definition.

SurveysDSL is the name of the modelling language defined for the production of online surveys. Figure 6 illustrates the meta-model defined using the domain editor of the framework. It includes four classes defined using the *EClass* classifier, which represent elements of the DSL. Each class contains some properties defined using the *EAttribute* construct. The *SurveysMetamodel* class although it has no semantic meaning, it is essential in the definition since it acts as the *root* container of the model. It is interpreted by the framework as the graphical canvas of the generated editor, which contains the three domain meta-classes, *Page, Question and Answer*. These classes are interpreted as graphical nodes.

Semantically this means that each survey domain model defined contains several pages, questions and answers. The meta-model definition explicitly states this relationship via the *pages, questions and answers aggregation type* definitions. Additionally each class includes distinct properties, which are enumerated onto Table 2 showing their precise semantic meaning.

TABLE 2. PROPERTIES OF THE META-CLASSES.

Meta-class	*Property*	*Semantic meaning*
Page	name	The name of the survey's page.
	title	The title of the survey.
	description	Describes the survey's topic.
Question	name	The name of the question
	qcontext	The question content.
	index	The sequence onto the page.
Answer	name	The name of the answer.
	acontext	The answer to the question specified.
	index	The sequence in terms of the question.

Complementary to the aggregation relationships defined, are the association relationships that reveal relations between model elements. The *Page* meta-class requires to be bound by an *association relationship*, called *hasQuestions*, with the *Question* meta-class. This depicts particularly that each page of the survey is associated with a specific set of questions. Furthermore the *Question* meta-class is in-turn bound with an *association relationship*, named *hasAnswers*, to the *Answer* meta-class. This rationally points to the fact that any specific question has a specific set of answers logically associated to it. Both relationships are interpreted by the framework as graphical links that semantically associate model elements.

After the informal meta-model definition the next step involves the imposition of constraints to convert the DSL into a formal one. Suppose that two questions are specified in the model and are associated to a particular page. Initially the individual properties of each question must be set accordingly. Each question though requires having a distinct name and content. The meta-model definition does not define such rules and consequently incorrect domain models can be defined. Therefore OCL expressions must be stated explicitly to avoid the definition of erroneous domain models.

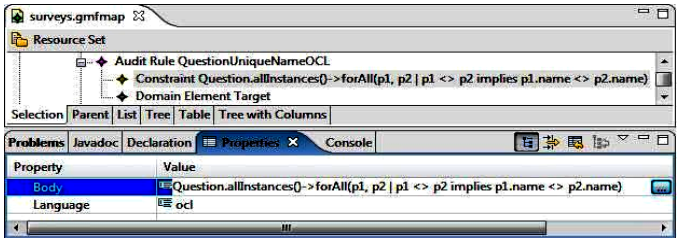

Fig. 7. Survey meta-model OCL constraints.

The *Constraint* on Figure 7 specifies that for all instances of the *Question* meta-class the *name* property must be different. As it is illustrated onto the figure the expression is included inside the *Audit Rule QuestionUniqueNameOCL*. The audit rule specifies particular properties, which the DSM environment of the SurveysDSL needs to consider when validating the models. An example property is the error message to be displayed in case the model does not satisfy that specific rule. Using the same procedure each audit rule is defined, to impose required constraints onto the meta-model.

After the successful definition of the meta-model and the imposition of constraints the graphical and tooling meta-models are extracted directly from the semantic meta-model. Following the graphical and tooling meta-models are mapped with the semantic meta-model. From the mapping meta-model obtained the environment for the online surveys domain is automatically generated.

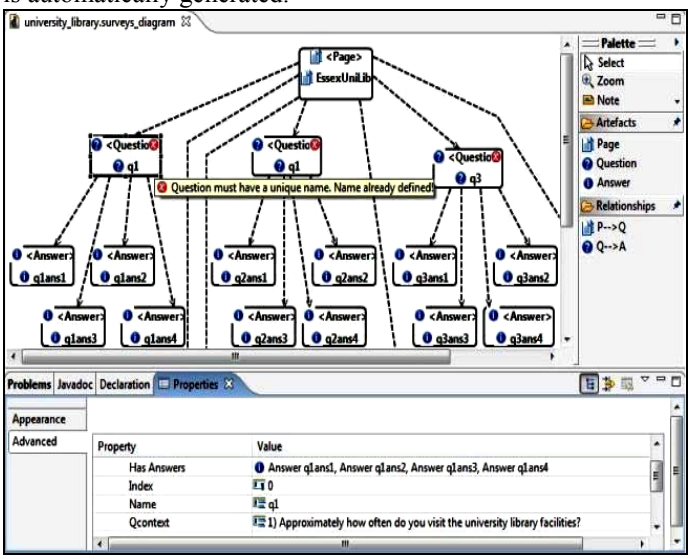

Fig. 8. University Library survey model.

Figure 8 illustrates part of the survey model investigated in this paper. The model is defined using the generated environment and subsequently the constraints are enforced to validate the model. As can be seen onto Figure 8 there is indeed a constraint violation since *Question q1* and *Question q2* were deliberately set to the same name; both *q1*. Therefore in order to precede the model needs to be refined and validated again to ensure that no more inconsistencies are present.

```
«FOREACH hasQuestions AS question»
 public void createQuestion«question.name»(){
        «question.name» = new JLabel("«question.qcontext»");
        «FOREACH question.hasAnswers AS answer»
        «answer.name» = new JRadioButton("«answer.acontext»");
        «answer.name».setActionCommand("«answer.name»");
        «answer.name».addActionListener(this);
        «ENDFOREACH»
        group«question.name» = new ButtonGroup();
        «FOREACH question.hasAnswers AS answer»
        group«question.name».add(«answer.name»);
        «ENDFOREACH»
 } «ENDFOREACH»
```

Fig. 9. Extract of the Java template-based code generator.

As soon as the model is validated the service implementation phase is undertaken. Using the framework's template editor, two distinct template-based code generators were defined. The primary generator aims at interpreting the model and generating a Java based implementation of the survey model. An extract of the generator is illustrated in Figure 9, which shows how each question and its containing answers are transformed to Java. For instance the *«**FOREACH** hasQuestions **AS** question»* statement, depicts that via the aggregation *hasQuestion* all the instances of the *Question* meta-class are accessed. In this way the properties of the meta-class are read and interpreted to the corresponding

Java statements. The equivalent is valid for the *«**FOREACH** question.hasAnswers **AS** answer»* statement.

The second template based generator created, interprets the model and generates an implementation for the Java 2 Micro Edition (J2ME) application platform. The two generated implementations attest to the fact that the generic framework is kept distinct from technological complexities. These are only introduced when the code generators are defined for generating the implementation from the abstract models.

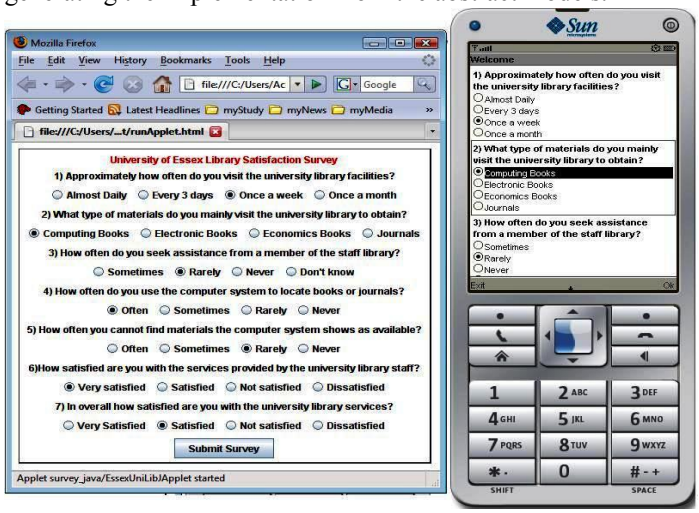

Fig. 10. Generated services running on diverse platforms.

Figure 10 illustrates the two services generated running on diverse execution platforms. The service implementations were generated and then deployed onto a mobile emulator device that uses the J2ME platform and onto a laptop device that has the Java virtual environment installed. Both implementations were generated from the same survey model illustrated previously onto Figure 8.

VI. Conclusion and Future Work

In this paper we have presented a model-driven software engineering technique and utilised a generic meta-framework to facilitate service creation. The framework has pulled together, under the Eclipse platform, the key artefacts necessary for generating service creation environments. The effectiveness of the framework has been showcased and validated via a domain-specific scenario for the development of an online survey system. Furthermore the application of the major components of the proposed framework illustrated the creation of an example survey.

Our aim as part of future work is to make use of the framework to define and generate a service creation environment for the pervasive services domain. The environment will include distinct languages for modelling service data, context-information, behaviour and user interactions. This will provide the capability to unambiguously model pervasive services, validate them and generate boilerplate service code for diverse platforms. Furthermore support for automatically creating test cases can be provided to enable ancillary service verification. Finally generation capabilities for other implementation technologies such as C++ or C# will also be introduced.

Acknowledgment

The work presented in this paper is partly supported by British Telecom under the Model-driven Component-based Systems Engineering (MOSE) project and the UK Engineering and Physical Sciences Research Council (EPSRC) under project PANDA (Policy-based Model-driven Pervasive Service Creation and Adaptation).

References

[1] D. X. Adamopoulos, G. Pavlou and C. A Papandreou, "Advanced Service Creation Using Distributed Object Technology", *IEEE Communications Magazine*, vol. 40, No. 3, March 2002.

[2] R. H. Glitho, F. Khendek and A. De Marco, "Creating value added services in Internet Telephony: An overview and a case study on a high-level service creation environment", *IEEE Transactions on System, Man and Cybernetics- Part C: Applications and Reviews*, vol. 33, No. 4, November 2003.

[3] N. Iscoe, G. B. Williams and G. Arango, "Domain modelling for Software Engineering", *IEEE International Conference on Software Engineering*, 1991.

[4] J. Evermann and Y. Wand, "Towards Formalizing Domain Modelling Semantics in Language Syntax", *IEEE Transactions on Software Engineering*, vol. 31, No.1, January 2005.

[5] Model Driven Architecture (MDA), Specification Guide v1.0.1, Object Management Group (OMG), [Online] Available: (June 2003), http://www.omg.org/docs/omg/03-06-01.pdf

[6] A. Kleppe, J. Warmer and W. Bast, *MDA Explained: The Model Driven Architecture: Practice and Promise*, Addison-Wesley, May 2005.

[7] D. S. Frankel, *Model Driven Architecture: Applying MDA to Enterprise Computing*, Wiley Publishing Inc., 2003.

[8] H. Wada and J. Suzuki, "A Domain Specific Modelling Framework for Secure Network Applications", *IEEE International Computer Software and Applications Conference (COMPSAC),* September 2006.

[9] B. Graaf and A. V. Deursen, "Visualization of Domain-Specific Modelling Languages Using UML", *IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, March 2007.

[10] A. V. Deursen, P. Klint and J. Visser, *Domain Specific Languages: An Annotated Bibliography*, Project DSL, Dutch Telematica Instituut.

[11] S. A. Thibault, R. Marlet and C. Consel, "Domain Specific Languages: From Design to Implementation Application to Video Device Drivers Generation", *IEEE Transactions on Software Engineering*, vol. 25, No. 3, May/June 1999.

[12] B. Steffen, T. Margaria, A. Claßen, V. Braun, M. Reitenspieß, "A Constraint-Oriented Service Creation Environment", *International Conference on Practical Application of Constraint Technology,* pp. 283-298, April 1996.

[13] J. Lennox, J. Rosenberg and H. Schulzrinne, Common Gateway Interface for SIP, RFC 3050, Internet Engineering Task Force, [Online] Available: (January 2001), http://www.ietf.org/rfc/rfc3050.txt

[14] J. Lennox and H. Schulzrinne, Call Processing Language Framework and Requirements, RFC 2824, Internet Engineering Task Force, [Online] Available: (May 2000), http://www.ietf.org/rfc/rfc2824.txt

[15] J.-L. Bakker and R. Jain, "Next generation service creation using XML scripting language", *in Proceedings of The IEEE International Conference on Communications*, 2002.

[16] Eclipse Foundation Inc., Eclipse Modelling Framework (EMF), [Online] Available: (2008) http://www.eclipse.org/emf/

[17] Eclipse Foundation Inc., Graphical Modelling Framework (GMF), [Online] Available: (2008) http://www.eclipse.org/gmf/

[18] INRIA Research Institution, Atlas Transformation Language (ATL), [Online] Available: (2008) http://www.eclipse.org/m2m/atl

[19] openArchitectureWare.org, openArchitectureWare (oAW), [Online] Available: (2008) http://www.eclipse.org/gmt/oaw

[20] Meta Object Facility (MOF) Core Specification, version 2.0, Object Management Group (OMG), [Online] Available: (January 2006) http://www.omg.org/docs/formal/06-01-01.pdf

[21] Object Constraint Language (OCL) Specification, version 2.0, Object Management Group (OMG), [Online] Available: (June 2006) http://www.omg.org/docs/formal/06-05-01.pdf