Architectural Variability Management in Multi-Layer Web Applications Through Feature Models

Jose Garcia-Alonso
Centro Universitario de Merida
Universidad de Extremadura
Avda. Santa Teresa de Jornet, 38
Merida, Spain
jgaralo@unex.es

Javier Berrocal Olmeda
Escuela Politecnica
Universidad de Extremadura
Avda. de la Universidad, s/n
Caceres, Spain
jberolm@unex.es

Juan Manuel Murillo
Escuela Politecnica
Universidad de Extremadura
Avda. de la Universidad, s/n
Caceres, Spain
juanmamu@unex.es

ABSTRACT

The development of large web applications has focused on the use of increasingly complex architectures based on the layer architectural pattern and different development frameworks. Many techniques have been proposed to deal with this increasing complexity, mostly in the field of model-based development which abstracts the architects and designers from the architectural and technological complexities. However, these techniques do not take into account the great variability of these architectures, and therefore limit the architectural options available for their users. We here describe a feature model that captures the architectural and technological variability of multilayer applications. Using this feature model as the core of a model-driven development process, we are able to incorporate architectural and technological variability into the model-based development of multilayer applications. This approach keeps complexity under control whilst flexibility on choosing technologies is not penalized

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures; D.2.13 [**Software Engineering**]: Reusable Software

General Terms

Design

Keywords

Multilayer architectures, feature model, design patterns, development frameworks, model-driven development

1. INTRODUCTION

Advances in technology as well as in the use of Internet by the general population have led to ever more advanced Web applications being built. The increased complexity of these applications naturally involves greater complexity in their development. One of the principal tools that software engineers have when confronting a complex development is represented by design patterns [11] – reusable general solutions for common problems in a given context.

In the context of Web applications, one of the commonest architectural patterns is the layer pattern [5]. This pattern proposes subdividing a system into a set of layers, each of which provides services to the layer above and consumes services from the layer below. Based on this principle, complex multilayer architectures can be developed in the way that a given layer can provide services to several upper-level layers. For instance, a business logic layer can provide services to a user interface layer and a Web service layer, or a single layer can be located transversally, and thus provide services to all the other layers, an example being a log layer.

Using the layer architectural pattern, the development is divided into several layers of a lower order of complexity than that of the complete system. The development of each of these layers presents its own challenges, however, so that it is usual to go back to using other design patterns. In order to facilitate the use of those design patterns, development frameworks have become especially relevant. This is reflected in the large number of existing frameworks [25], the traffic generated by their mailing lists, and the number of job offers requesting experience in their use [23]. These frameworks "provide domain-specific concepts, which are generic units of functionality. Developers create framework-based applications by writing framework completion code, which instantiates these concepts" [3].

The use of the layer architectural pattern combined with specific development frameworks for each of the layers in the application provides greater quality in terms of reliability and maintainability [5]. But it also introduces new problems [14]. Everyone involved in building such an application must have in-depth technical knowledge in the use of the chosen frameworks. Frameworks are advanced tools whose use requires highly trained personnel. This problem can be mitigated using model-driven development techniques that abstract developers from the technical details of the technology [24].

However, current solutions are not fully optimal due to its limitations [16]. In an environment with many rapidly evolving technologies, model-driven techniques need continual updates to keep pace with technological advances. These

FOSD'12 September 24-25, 2012, Dresden, Germany

updates must be performed by staff with profound knowledge in three areas: the new technology to be included, the transformation language used, and the model-driven development technique or tool into which it will be incorporated. This makes it too complicated to upgrade existing model-driven techniques to cover the large number of technologies available.

To address these issues, we here present a feature model that captures the architectural and technological variability of multilayer Web applications. This model contains the commonest layers of these applications, the design patterns used in each layer, and the development frameworks that implement those patterns. Its use facilitates the architect's work in defining the software architecture that best meets the system requirements, and in choosing the most suitable technology for the implementation.

The main advantage of this model is that it can be used as the core of a model-driven development process, as some of the currently available. This allows the developers to be largely abstracted from the technological details, an aspect that is especially important in rapidly evolving environments such as that of framework-based development, and keep complexity under control while increasing the flexibility of these processes.

The rest of the paper is organized as follows. The second section presents the background for this study. The third section describes the process used to obtain the feature model. The fourth section details this feature model and how it can be expanded to capture new or evolved technologies. The fifth section shows how the feature model can be used as the core element of a model-driven development process. The sixth section describes related work. And finally, the seventh section contains the conclusions of the study.

2. BACKGROUND

The techniques proposed by researchers in the areas of Web engineering and model-driven development have led to major advances in simplifying the work of the developers of multilayer Web applications. The use of various model-to-model transformations and code generation techniques allows these applications to be developed without the need for extensive knowledge of the technologies being used.

Nevertheless, although these techniques achieve the objective of facilitating software architect's work, they have the drawback of limiting his or her options in designing the application's architecture or choosing the technologies to use in the implementation. This is because these techniques impose the use of specific software architectures inherent to the models needed for the development, and the architect is not allowed to adapt them to the system's requirements. Moreover, the architect can only choose from among a very limited set of technologies to use in the implementation. In many cases, the system requirements or external constraints to the development mean that such limitations are unacceptable.

Some of the most important of these techniques and how they limit the software architect's options are the following:

- In [1], Acerbis et al. describe a modeling language, WebML, and a tool, WebRatio, which cover the entire process of Web application development. The work is based on the conceptual modeling of the application in two design stages: data design with which to organize the application's information objects, and hypertext design defining the application's interface given the preceding data design. In this way, the application is divided into two basic layers –persistence and presentation– regardless of its actual requirements. Also, the architect cannot modify this structure. Moreover, the code generated from these designs is based on ANT, XSLT, or Groovy technologies, so that the architect can not choose from the wide range of other existing technologies.

- In [17], UWE (UMLbased Web Engineering), a method of model-driven Web application development based on UML, is defined. The process proposed in UWE is based on creating, in accordance with the application's requirements, a number of models of content, navigation, processing, presentation, and adaptability. As in the previous case, these models inevitably restrict the application's architecture. To address this problem, the authors propose using an additional technique called WebSA which we shall discuss below in the Related Work section. The code generation capabilities offered by this process use a specific framework (Spring), and the authors specifically observe that it is possible to generate code for other technologies if the corresponding transformation rules are previously available. However, although this is a valid solution with which to address the issue of ongoing technological evolution, it requires highly trained staff in both the technology to include and the techniques used to perform the transformations.

- Some studies, such as [28], increase the capacity of previous proposals in such areas as the development of RIAs (Rich Internet Applications). However, none of these improvements expands the architect's options in designing the architecture of the application or choosing the technologies to use in development.

In some cases, the limitations of these techniques are unacceptable to some clients or projects with specific requirements of technological flexibility, such as the integration of legacy systems or in cases when certain quality factors such as high performance or stability are needed, restricting the use of such proposals. Since application architectures must conform to the corresponding requirements in the best way possible, they can not use architectures imposed by external systems. Moreover, the technologies to use in the development are often imposed by the customer's requirements or by company policy, so that the use of a technique that imposes a very limited set of development frameworks is not really an option.

We shall here present a feature model that captures the architectonic and technological variability of multilayer applications. This feature model can be used as the core of a set of transformations that allow the architect to obtain a specific design adapted to whichever architecture and technologies are selected in accordance with the requirements of the application being developed.

3. OBTAINING THE FEATURE MODEL

The main aim pursued in the creation of this model is to incorporate architectural and technological variability into

Table 1: Essential framework information.

Framework	Layer	Design Patterns	Implementation techniques
Axis	Web services	SOAP	NA
CXF	Web services	REST	NA
DWR	Presentation	Web remoting	NA
		Page rearrangement	NA
Hibernate	Persistence	DAO	JPA
			XML
			Annotations
Ibatis	Persistence	DAO	NA
JDBC	Persistence	DAO	NA
JSF	Presentation	MVC	NA
		Web remoting	NA
		Page rearrangement	NA
jUnit	Test	xUnit	NA
Log4j	Log	Logger	NA
PicoContainer	Business logic	IoC	NA
Spring	Business logic	IoC	XML
			Annotations
SpringSecurity	Security	Authentication	LDAP
			OpenID
			JaaS
			HTTP
		Authorization	Web request
			Method invocation
			Access to instances
SpringWS	Web services	REST	NA
Struts	Presentation	MVC	NA

the development of multilayer applications. Feature modeling [15] is one of the more widely accepted techniques for variability modeling [26]. In particular, our proposal uses the Cardinality Based Feature Modeling technique described in [9] since (i) it is one of the most extensively used, (ii) it is of proven utility in working with development frameworks [3], and (iii) it meets all the requirements we set out in undertaking this study. Nonetheless, it may readily be replaced by another variability modeling technique.

Since the intention was to use the feature model as the core of a model-driven development process, it had to have a well-defined structure or conform to some kind of "metamodel" so as to be treated automatically later. Such structure, however, had to be flexible enough to incorporate the large number of existing technologies. It also had to be capable of incorporating both any new technology that might arise and the evolution of existing technologies. Indeed, this was the main criterion imposed on the creation of the feature model.

To obtain a feature model that meets these requirements, we followed a bottom-up strategy. In particular, we studied a large number of development frameworks in order to extract concepts that would form the structure or "metamodel" of the feature model. For the structure to be as flexible as possible, more than 10 Java development frameworks were chosen from different developers and with different roles and goals. These frameworks were selected for being among the most commonly used within their scope [25, 23, 31]. Following is the lists of frameworks analysed: Axis, CXF, DWR, Hibernate, Ibatis, JDBC, JSF, jUnit, Log4j, PicoContainer, Spring, SpringSecurity, SpringWS and Struts.

The first architectural decision to be taken when building a multilayer application is to determine the layers of which it will be composed. Therefore, the first criterion used in analysing the development frameworks was to determine the layer or layers in which they are used.

After determining the layers that make up the application, the architect must define the design patterns to be used in implementing each of them. In particular, knowledge of which design patterns a development framework supports is of particular importance at this stage.

Finally, a framework may allow different kinds of implementation for the same design pattern. If one does not want to lose the advantages offered by the development frameworks, these different implementation techniques need to be taken into account in the feature model.

Given these considerations, we studied the frameworks listed above. The information extracted from that analysis is summarized in Table 1.

Starting from this information, we extracted the structure that the feature model would need to have. Figure 1 shows a feature model with that structure.

In general, the scope of all the frameworks studied was in a single layer. Even when the same developer supports multiple layers, this is usually implemented by being distributed among different frameworks that can be used independently. For example, the frameworks Spring, Spring Security, and SpringWS belong to the same developer, but each targets a single layer and is treated as a separate product. This, together with the fact that the layer is the main architectural pattern applied in the development of multilayer applications, the most appropriate would be for the first level of the feature model to consist of the possible layers of which an application may be composed.

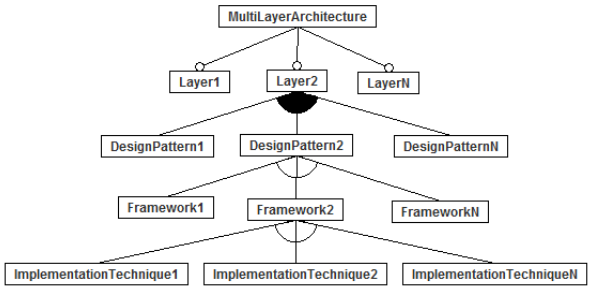

Figure 1: Feature model structure.

As mentioned above, one or more design patterns are used to simplify the implementation of each layer. Usually these design patterns are specific to each layer. Therefore, it would be appropriate for each layer present in the feature model to include the group of features representing the design patterns that can be used.

Usually development frameworks provide support for one or more design patterns. Therefore, each design pattern included in the feature model must specify the frameworks that can be used to implement it. This may result in the same framework appearing more than once in the feature model, provided that this framework supports several design patterns. This poses no problem, since the occurrence of a framework in the model implies that the framework can be used in the implementation of a particular pattern, but it does not imply that its use for a specific pattern requires the use of the same framework in all the patterns of a given layer. For example, the JSF framework supports the implementation of three design patterns – MVC, web remoting, and page rearrangement. This implies that the framework will appear thrice in the feature model. Nonetheless, the use of JSF to implement the MVC pattern in a particular application does not imply that other frameworks can not be used for the other patterns.

Finally, it is common for a framework to provide different techniques for implementing a design pattern. These techniques generally vary in the syntax used, but end up providing the same results. An example of this is dependency injection in the Spring framework. This can be done using Java annotations or using an XML configuration file. This variability aspect was also taken into account in the feature model. Where applicable, the feature model offers the techniques supported by a framework in the implementation of a given design pattern.

4. ARCHITECTURAL AND TECHNOLOGICAL FEATURE MODEL

Based on the information obtained during the framework analysis (Table 1) and the structure shown in Figure 1, we constructed a feature model that captures the variability of the frameworks considered. A fragment of that model is shown in Figure 2

This model can not be created solely with the information presented in the previous section, however. For the model to be representative enough for use in a development process, it must contain information not only about the frameworks but also about their interrelationships. This is because the frameworks chosen for an application development process, while independent entities, must interact and communicate with each other.

This implies that, in choosing the frameworks to be used in the development of an application, their possible incompatibilities need to be explored, and this information needs to be incorporated into the feature model. One way to include this information in the model is as constraints.

We chose OCL [21] as the language to introduce such constraints into the feature model because it has previously been used for this purpose [10], and because it is particularly well-suited to use in a model-driven process such as that which we shall be presenting in the next section.

With these constraints, we can express situations like that of when using some given framework this will prevent the use of certain other frameworks. This may be because there is no possibility of communication between them, or it may be that they are incompatible by design. For example, in the model shown in Figure 2, one can choose the Struts framework to implement the MVC pattern in the persistence layer. At that point, the JSF framework would not be a suitable choice for the implementation of any of the design patterns in the same layer. This is due to JSF being an MVC focused framework which provides only secondary support to the implementation of the other patterns. While this might not create any strict incompatibility between frameworks, it greatly complicates the communication between frameworks if JSF is not used to implement the MVC pattern. This constraint is expressed as follows in OCL (detailed information about how to express constraints in OCL can be found in [8]):

```
context Presentation inv:
MVC.Struts.isDefined() implies
   not(WebRemoting.JSF.isDefined())
   and not(PageRearrangement.JSF.isDefined())
```

Similarly constraints expressing the obligation to use a specific framework can be added to the feature model. Such constraints may occur when using a framework in a specific pattern that requires the use of another framework in another pattern. This may also be due to compatibility issues when a framework is compatible only with a limited set of others, or it may merely be for convenience. For example, in the model shown in Figure 2, a constraint could be included to specify that if the JSF framework is chosen for the MVC pattern then the same framework must also be chosen to implement the remaining patterns of the same layer. Constraints such as the one presented immediately below can also be included to indicate that if the SpringWS framework is chosen for implementing Web services following the REST pattern, the Spring framework must then be chosen for dependency injection. Again this is not due to any strict incompatibility between frameworks, but to the combination of these two greatly simplifying the communication between layers.

```
context MultiTierArchitecture inv:
WebServices.REST.Spring_WS.isDefined() implies
   InversionofControl.DependencyInjection.Spring
       .isDefined()
```

Aditionally, the OCL constrains could be used to include metrics about two frameworks integration level or to include qualitative aspects like the performance indicators of a framework.

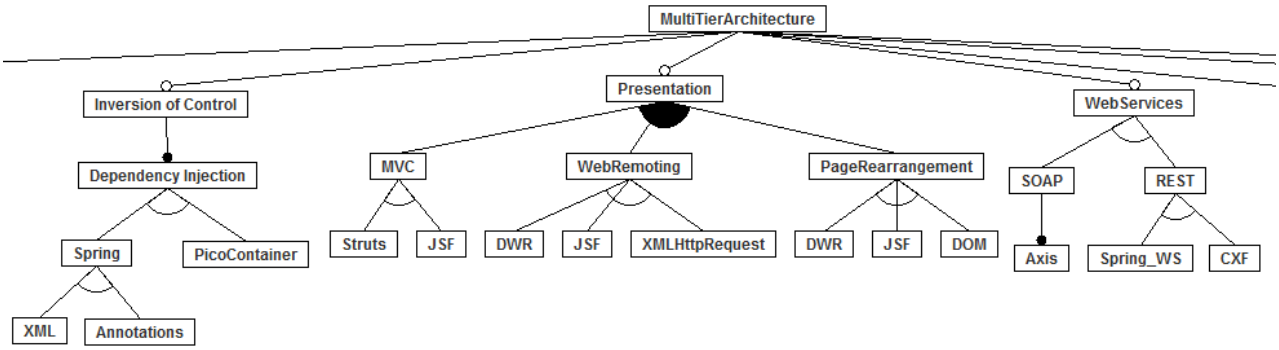

Figure 2: Excerpt of the feature model.

With the addition of all the necessary constraints, the model incorporates all the information needed for use in a model-driven development process. In particular, one has a feature model that captures the architectural and technological variability of multilayer applications.

In view of how rapidly development frameworks evolve, one of our main concerns in constructing this model was its ease of extension to adopt new or evolved technologies. To include a new technology or a new version of an existing technology in the model, it is sufficient to include the feature for that technology in the design patterns that it supports, and to study its relationships with the other frameworks so as to add the corresponding constraints. If the new technology to be adopted requires the inclusion of a new pattern or layer, this must be added to the model before including the technology.

Additionally, the use of OCL constraints endows the model with greater flexibility. These constraints can be used not only to express the relationships between frameworks, but also to include in the model much information about the multilayer architectures. A clear example is the use of OCL constraints to specify the internal policies of a particular development company. For example, the company may have designed a standard architecture that it uses to develop most of its projects. These constraints can be used to strengthen the implementation of such an architecture, or even to reinforce the specific ways in which a framework is used to make this use uniform over all of the company's projects.

5. A FEATURE MODEL AS THE CORE OF A MDD PROCESS

The feature model presented in the previous section is interesting in itself as a taxonomy of a set of technologies used in multilayer application development. The advantages provided by this model are really exploited, however, when it is used as the centrepiece of a model-based development process. In this section we shall discuss how the feature model, in combination with various modern model-driven techniques, can be used to guide the development process.

In this process, the software architects and designers configure the feature model in stages. "Each stage takes a feature model and yields a specialized feature model, where the set of systems described by the specialized model is a subset of the systems described by the feature model to be specialized" [9]. In this configuration process, the layers to be included in the development of the application are selected, then the design patterns used in each layer, and finally the frameworks used for the development of each pattern and the form in which they will be used.

This configuration process can be done by the architect and designers based on their expertise in the technologies involved, or it can be assisted. Such assistance is based on the use of a technique described by the present authors in [7]. This technique facilitates the choice of the patterns that best meet the non-functional requirements of the application being developed, and offers architects and designers a possible configuration of the feature model that matches the application's requirements.

At each step in the staged configuration of the feature model, the initial design of the application is further refined in accordance with any architectural decisions taken. This refinement of the initial design is performed using model transformations. However, the rapid evolution of technologies makes it inefficient to use technology specific transformations. Instead, different model-driven techniques allow one to write transformations (e.g., high-order transformations [27] or variable transformations [29]) that can be adapted to the evolution of the technologies involved.

Performing these transformations requires additional information. Specifically, information is needed about the relationship between elements of the initial design and the selected features in the feature model. It is not always possible to obtain this information in the same way as is done to obtain information to assist in the configuration of the feature model, especially when different frameworks are used to implement patterns in the same layer.

In such cases, it is necessary to have a mechanism that allows architects and developers to specify which features should be applied to each element of the initial design of the application. This operation can be performed using the technique proposed by Arboured et al. in [4]. Their technique enables features to be related to specific elements of a model, in this case to specific elements of the initial design. Other techniques to relate feature models with other kinds of model are that represented by the Clafer language [6] and those discussed by Voelter and Visser [30]. Clafer allows metamodeling and feature modeling to be combined into a single language, and Voelter and Visser study the application of domain-specific languages in product-line engineering.

Finally, the specialization of the initial design of an application into a specific design for the chosen architecture

and technologies is carried out, as mentioned above, using model transformations. Feature models, however, are designed for use in product lines, and are not readily applied to processes that involve model transformations specific to model-based development. In particular, the QVT language [22], the OMG standard for performing model transformations, requires the two models involved in a transformation to be based on MOF metamodels, but feature models are not based on a metamodel defined using this system.

Besides, these models would present another problem regardless of whether or not they were based on a metamodel suitable for the realization of the transformations: the feature model is not used just as it is, but different configurations of the model are also necessary. This increases the difficulty of using these models in transformations because the configurations are at a lower level in the four-layer architecture of MOF.

These problems have been resolved by Gomez and Ramos [12]. They define a metamodel for modeling CBFM feature models based on MOF. The model shown in Figure 2 can thus be converted to a model based on their metamodel. Additionally, Gomez and Ramos provide a set of QVT transformations that convert the feature model into an MOF-based metamodel. This metamodel allows configurations of the feature model to be created at the appropriate level of MOF for their use in transformation processes. A feature model based on MOF also facilitates the use of OCL constraints such as those discussed in the previous section.

The combination of the feature model presented here with the various techniques that have been mentioned in this section enables this model to be used as the core of a model-driven development process. This process simplifies the conversion of an initial design of the application (independently of which architecture and technologies are going to be used in the implementation) into a specific design for the chosen architecture and technologies. A major advantage of this process in an environment in which development frameworks evolve so rapidly is that it needs no in-depth knowledge of the technical operation of the frameworks.

6. RELATED WORK

As was noted above, there have been numerous works in the area of model-driven development which deal with the development of enterprise applications that have complex architectures, and are implemented using development frameworks. This is especially so in Web engineering, with such studies as WebML [1], UWE [17], RUX [28], etc.

These studies all provide techniques with which to richly model the functionality of Web applications, and a set of transformations that allow the user to generate the application's code from those models. They do not, however, provide much flexibility. In most cases, applications developed with these techniques have a fixed architecture which can be neither influenced nor altered by the software architect, and which is often implicit in the models that are created. For example, most of these methods require both a persistence model and a presentation model, thus forcing the presence of these layers in any application which is developed. The case is similar in the choice of technology. The use of these methods requires the adoption of technologies for which the transformations provided in the method can generate the application's code, thus further limiting the choices available.

Nevertheless, in this same field of Web engineering there have been proposals aimed at endowing the process of designing the application's architecture with a certain flexibility. Melia and Gomez [18] propose an extension of the methods mentioned above which they call Web Software Architecture (WebSA). This extension adds flexibility to the previous methods by providing them with the means required to define the architecture used. Two models are added to achieve this goal: a subsystems model and a configuration model. These two models define the architectural features of the Web application that is to be developed. They are both linked to the functional models of the application (which may be constructed with any of the previous methods: OO-H, WebML, etc.), and are used to generate the application with the desired architecture through a series of transformations.

A more recent study by those same authors [19] describes a similar proposal aimed at the development of RIAs. In particular, it proposes OOH4RIA as an OOH extension specifically designed for the development of RIAs. The authors describe the use of a feature model, similar to the one used in the present work, to define RIA features, and a component model to explicitly represent the architecture of the RIA. Once again, these models are used together with the functional models of OOH to generate the application's source code.

These two works (especially the more recent) are closely related to that presented here. They pursue the same goal: to enable developers who use model-driven techniques to influence the architecture of their applications. However, the main focus of the works of Melia and Gomez is on RIA, whereas the work presented here is intended to provide support for all applications that use a multilayer architecture. Also, in contrast with the technique presented here, they do not allow developers to decide which technologies will be used for the development of the application.

In the field of framework-based software development, the studies of Antkiewicz et al. [3] and Heydarnoori et al. [13] are of particular interest. Antkiewicz et al. propose techniques that allow framework-specific designs to be modeled, and these designs then to be used to generate the source code. Heydarnoori et al. propose a technique to automatically extract framework concept-implementation templates. Unlike the work proposed here, these proposals are centred at a low level of abstraction, thus requiring their users to possess a deeper technical understanding of the frameworks they want to use. The present work raises the level of abstraction by starting from the design of the software architecture.

Finally, some works in the architecture quality field have proposed techniques to increase a system's architectural variability. In [20], Naab and Stamel propose a technique to achieve flexibility during the architecture design stage for this flexibility to be exploitable during system evolution. This technique improves variability in the architecture, so that it can be adapted to future evolution of the system, while the technique presented here is designed to provide as much variability as possible to the architect before construction begins. In [2], Alebrahim and Heisel present a UML-based approach to modeling variability in architectures by adopting the notion of feature modeling. In their proposal, the variability is that introduced into an architecture by quality attributes, to take into account how they affect the

final architecture. In our proposal, quality attributes are used (as described in [7]) to guide the configuration process of the architectural feature model.

To the best of the authors' knowledge, there has been no previous work on using product lines and model-driven development techniques to increase the options of software architects in the work of selecting which design patterns and development frameworks to use in the development of a multilayer application.

7. CONCLUSIONS

We have presented a feature model that captures the architectural and technological variability of multilayer applications. This model was obtained from the study of a large number of technologies. We also presented a model-driven development process which uses the feature model as its core. This model pretends to significantly increase the options available to developers when using a model-driven process to develop a multilayer application.

The following phase in this research line will be to define a method with which to incorporate into the development process the technical details of using the technologies described in the feature model. Such a model would allow one to choose the frameworks that are going to be used for a particular development. The technical details of its use, however, would have to be covered by developers who are experts in those technologies, or by transformations created by experts in those same technologies who also have advanced knowledge about transformation techniques. This situation clearly greatly complicates the incorporation of new technologies into the model-based development processes used today.

8. ACKNOWLEDGMENTS

This research was supported by the Spanish Ministry of Science and Innovation under Project TIN2011-24278, and by the Spanish Centre for Industrial Technological Development under Project GEPRODIST.

9. REFERENCES

[1] R. Acerbis, A. Bongio, M. Brambilla, S. Butti, S. Ceri, and P. Fraternali. Web applications design and development with webml and webratio 5.0. In R. F. Paige and B. Meyer, editors, *TOOLS (46)*, volume 11 of *Lecture Notes in Business Information Processing*, pages 392–411. Springer, 2008.

[2] A. Alebrahim and M. Heisel. Supporting quality-driven design decisions by modeling variability. In *8th ACM SigSoft International Conference on the Quality of Software Architectures*, pages 43 – 48, june 2012.

[3] M. Antkiewicz, K. Czarnecki, and M. Stephan. Engineering of framework-specific modeling languages. *IEEE Trans. Software Eng.*, 35(6):795–824, 2009.

[4] H. Arboleda, R. Casallas, and J.-C. Royer. Dealing with fine-grained configurations in model-driven spls. In D. Muthig and J. D. McGregor, editors, *SPLC*, volume 446 of *ACM International Conference Proceeding Series*, pages 1–10. ACM, 2009.

[5] P. Avgeriou and U. Zdun. Architectural patterns revisited - a pattern language. In A. Longshaw and U. Zdun, editors, *EuroPLoP*, pages 431–470. UVK - Universitaetsverlag Konstanz, 2005.

[6] K. Bak, K. Czarnecki, and A. Wasowski. Feature and meta-models in clafer: Mixed, specialized, and coupled. In B. A. Malloy, S. Staab, and M. van den Brand, editors, *SLE*, volume 6563 of *Lecture Notes in Computer Science*, pages 102–122. Springer, 2010.

[7] J. Berrocal, J. García-Alonso, and J. M. Murillo. Facilitating the selection of architectural patterns by means of a marked requirements model. In M. A. Babar and I. Gorton, editors, *ECSA*, volume 6285 of *Lecture Notes in Computer Science*, pages 384–391. Springer, 2010.

[8] J. Cabot and M. Gogolla. Object constraint language (ocl): A definitive guide. In M. Bernardo, V. Cortellessa, and A. Pierantonio, editors, *Formal Methods for Model-Driven Engineering*, volume 7320 of *Lecture Notes in Computer Science*, pages 58–90. Springer Berlin / Heidelberg, 2012.

[9] K. Czarnecki, S. Helsen, and U. W. Eisenecker. Staged configuration through specialization and multilevel configuration of feature models. *Software Process: Improvement and Practice*, 10(2):143–169, 2005.

[10] K. Czarnecki and P. Kim. Cardinality-Based Feature Modeling and Constraints: A Progress Report. In *Proceedings of the International Workshop on Software Factories at OOPSLA 2005*, 2005.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, USA, 1995.

[12] A. Gómez and I. Ramos. Cardinality-based feature modeling and model-driven engineering: Fitting them together. In D. Benavides, D. S. Batory, and P. Grünbacher, editors, *VaMoS*, volume 37 of *ICB-Research Report*, pages 61–68. Universität Duisburg-Essen, 2010.

[13] A. Heydarnoori, K. Czarnecki, and T. Tonelli Bartolomei. Two studies of framework-usage templates extracted from dynamic traces. *IEEE Transactions on Software Engineering*, 2011.

[14] D. Hou and L. Li. Obstacles in using frameworks and apis: An exploratory study of programmers' newsgroup discussions. In *ICPC*, pages 91–100. IEEE Computer Society, 2011.

[15] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute, November 1990.

[16] N. Koch, S. Melia-Beigbeder, and J. Vara-Mesa. Model-driven web engineering. *Upgrade, the European Journal for the Informatics Professional*, 9(2):40–45, 2008.

[17] A. Kraus, A. Knapp, and N. Koch. Model-driven generation of web applications in uwe. In N. Koch, A. Vallecillo, and G.-J. Houben, editors, *MDWE*, volume 261 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[18] S. Meliá and J. Gómez. The websa approach: Applying model driven engineering to web applications. *J. Web Eng.*, 5(2):121–149, 2006.

[19] S. Meliá, J. Gómez, S. Pérez, and O. Díaz. Architectural and technological variability in rich internet applications. *IEEE Internet Computing*, 14(3):24–32, 2010.

[20] M. Naab and J. Stammel. Architectural flexibility in a software-systemŠs life-cycle: Systematic construction and exploitation of flexibility. In *8th ACM SigSoft International Conference on the Quality of Software Architectures*, pages 13 – 22, june 2012.

[21] OMG. Object Constraint Language (OCL) 2.2, 2010.

[22] OMG. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, 2011.

[23] M. Raible. Comparing java web frameworks. `http://static.raibledesigns.com/repository/presentations/ComparingJavaWebFrameworks-ApacheConUS2007.pdf`, Apache convention, 2007.

[24] D. C. Schmidt. Guest editor's introduction: Model-driven engineering. *IEEE Computer*, 39(2):25–31, 2006.

[25] T. C. Shan and W. W. Hua. Taxonomy of java web application frameworks. *E-Business Engineering, IEEE International Conference on*, 0:378–385, 2006.

[26] M. Sinnema and S. Deelstra. Classifying variability modeling techniques. *Information and Software Technology*, 49(7):717 – 739, 2007.

[27] M. Tisi, F. Jouault, P. Fraternali, S. Ceri, and J. Bézivin. On the use of higher-order model transformations. In R. F. Paige, A. Hartman, and A. Rensink, editors, *ECMDA-FA*, volume 5562 of *Lecture Notes in Computer Science*, pages 18–33. Springer, 2009.

[28] M. L. Trigueros, J. C. Preciado, R. Morales-Chaparro, R. Rodríguez-Echeverría, and F. Sánchez-Figueroa. Automatic generation of rias using rux-tool and webratio. In M. Gaedke, M. Grossniklaus, and O. Díaz, editors, *ICWE*, volume 5648 of *Lecture Notes in Computer Science*, pages 501–504. Springer, 2009.

[29] M. Voelter and I. Groher. Handling Variability in Model Transformations and Generators. In *Proceedings of the 7th Workshop on Domain-Specific Modeling (DSM'07) at OOPSLA '07*, 2007.

[30] M. Voelter and E. Visser. Product line engineering using domain-specific languages. In *Software Product Line Conference (SPLC), 2011 15th International*, pages 70 –79, aug. 2011.

[31] T. Zimmer. Taxonomy for web-programming technologies. Master's thesis, UniversitÃČÂd't Koblenz-Landau, Campus Koblenz, 2012.