Architectural Decisions in the Development of Multi-Layer Applications

Jose Garcia-Alonso
Quercus Software Engineering Group
Centro Universitario de Merida
Merida, Spain
Email: jgaralo@unex.es

Javier Berrocal Olmeda
Juan Manuel Murillo
Quercus Software Engineering Group
Escuela Politecnica
Caceres, Spain
Email: {jberolm,juanmamu}@unex.es

***Abstract*—Multi-layer architectures have become one of the most widely used architectures for enterprise application development. Among other reasons, this is due to the proliferation of development frameworks simplifying the implementation of applications based on such architectures. However, the software architect is faced with a significant challenge at the beginning of the development process with having to decide among the great number of design patterns and development frameworks that support these architectures. The present work proposes a technique to assist the architect in deciding which technologies are best suited to satisfying both the functional and the non-functional requirements of the system. This technique forms part of a broader procedure to facilitate the software architect's task of converting a preliminar concept of an application into a specific design optimized to the project in hand.**

Keywords*—Multi-layer architectures; design patterns; development frameworks; architectural knowledge.

I. INTRODUCTION

A significant proportion of applications being developed today are targeted at enterprises. They tend to be complex systems with significant scalability and performance requirements. These requirements are further complicated by the rise in recent years of cloud computing and development for mobile platforms. When these applications make use of such environments the non-functional requirements regarding reliability, performance, integration, security, migratability, etc; gain even greater relevance [1].

The focus of the present study is on the development of the back end of these applications – specifically, of those whose development is based on the use of multi-layer architectures. Defining and designing the architecture of a system of this type is an arduous and complex process for the architect.

Firstly, many frameworks and design patterns have been proposed to simplify the implementation of these architectures [2]. Currently, the use of development frameworks, and consequently of the design patterns that they help to implement, is a widely extended practice. Proof of this is the large number of available frameworks [3], the number of versions released annually, and the job offers that require their skills [4]. The great number of existing design patterns and development frameworks forces architects to devote substantial effort to learning them. It is not enough to obtain an in-depth knowledge of a set of them, it is necessary to have adequate knowledge about all of them, the web of interactions between them [5] and the use of one or another favouring or penalizing the fulfilment of certain non-functional requirements.

And secondly, in order to make these decisions properly, the architect must have a thorough knowledge of the requirements and of the relations between them. The architect must extract the knowledge about the system requirements from the analysis of a series of documents on which, in many cases, the relationship between functional and non-functional requirements are not explicitly detailed [6]. The ability of the architecture to meet the system's requirements depends on the interpretation of these documents. Therefore, any misinterpretation on her part in this complex analysis implies the inclusion of errors in the architecture.

The combination of these two factors exposes the architect to situations in which a misinterpretation could lead to the choice of an inappropriate design pattern or development framework, with the problems that it would entail [7]. The present work focuses on the architect's decision making. Its principal contribution is a technique which makes use of a feature model to provide the architect with a catalogue of the commonest architectural decisions in the development of framework-based multi-layer applications [8]. The architect can use that catalogue, alongside the preliminary design of the application marked with quality attributes [9], as a basis for orderly decision-making. The decisions actually made by the architect are also recorded and later they can be used as design guidelines in developing similar applications [10].

The rest of this communication is organized as follows. Section 2 presents the motivations for this work. Section 3 gives a complete overview of the proposal. Section 4 details the proposed decision-making process and the automatisms provided. Section 5 specifies the tools which support this proposal. Section 6 gives a review of the most significant related work. Finally, Section 7 presents the conclusions to be drawn from the study, and some indications of future work planned in this line of research.

II. MOTIVATION

During the development of industrial software applications, the preliminary designs obtained from the requirements are not usually implemented as such. First, they must be adapted to the chosen multi-layer architecture [11].

Once the layer architectural pattern [12] has been applied to the initial design of a system, different design patterns may be used in each layer. For example, the Data Access Object (DAO) pattern can be used in the design of a persistence layer and the Model-View-Controller (MVC) pattern to design a presentation layer. This kind of multi-layer architecture has become widely

 ISBN: 978-1-61208-304-9

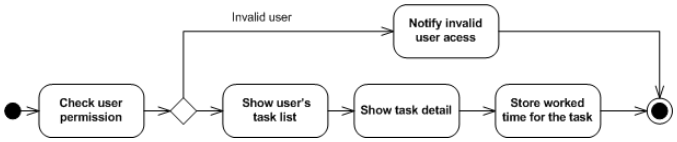

Fig. 1. Activity diagram (part of the initial design).

accepted in the industry, especially since the introduction of development frameworks [2].

However, the use of such architectures has its downsides. Specifically, what was once a clear advantage, nowadays, with the explosion in the number of frameworks and patterns, has become an additional risk. The architect needs a depth knowledge about a large number of frameworks and the interrelations between them. For this reason, the architect's work becomes more error prone, and, worse, these are errors that may have a significant impact on the overall project.

In order to motivate the problems addressed by this work we present here an example of the design process for an application's architecture. Figure 1 shows an activity diagram of a very common use case in enterprise applications. This use case allows the system's users to check a series of elements, to see detailed information about any one of them, and to modify that information. The system performs a check on whether or not the user has permission to perform that operation. If not, a notification is sent informing of an invalid access attempt.

Establishing the system requirements is the starting point for architects designing a new architecture. The designed architecture should maximize the chances of complying with all the requirements. This is in itself a complicated task. In many cases systems are asked to meet requirements that are difficult to combine and the architect should reach a balance [6].

Once the architect acquires all the information about the requirements, he or she should start its design. For this, the architect must take into account the large number of patterns available. Choosing a particular pattern can lead to different degrees of requirements being satisfied, especially in the case of non-functional requirements [13].

Referring the case study, the developed system must meet certain security and auditing requirements. If the architect omits, forgets or misinterprets the relation between this requirements, she might try to meet both requirements at the same time. However, these requirements may conflict making the architect choice a possible cause of future errors.

This study presents a technique to simplify and record architectural decisions in the development of multi-layer applications. Studies such as those of Zimmermann [10], [14] focus on the architectural decisions making process in a similar way as discussed in this article. However, to the best of the authors' knowledge, despite the industrial acceptance of multi-layer architectures and development frameworks, there has been no previous work on support for architectural decision making in framework based multi-layer applications.

This work forms part of a broader proposal that covers the entire process of designing these applications. In the next section, we shall briefly describe the complete proposal so as to provide a clear context for the contribution to be described in the rest of the paper.

III. Multi-layer Enterprise Applications

Figure 2 shows a complete diagram of the process proposed for the development of framework-based multi-layer applications.

It shows how the proposed process begins with the preliminar design, normally consisting of a use case diagram and multiple activity diagrams representing the behaviour of those use cases. In activity 1 this design has to be refined by the architect or requirements experts to include information about the quality attributes of the system.

As mentioned above, usually the relationship between functional and non-functional requirements are not explicitly detailed [6]. To make these relationships explicit, the architect or the requirements expert mark the preliminary design with information about the quality attributes to be met by the application. The technique used to accomplish this marking is described in more detail in another paper by the authors [9].

Once the architect has the marked design, the next task is to select the layers into which to split the application, activity 2 in the diagram. In order to simplify this task, the process offers to the architect an initial selection of layers. This initial selection is based on the preliminary design and the information added by the marks. However, is the architect who must refine, validate or reject it based on other criteria such as technological limitations, type of project, client, etc. This task is done in the activity 3 in the diagram.

Once the layers have been selected, the initial design can be refined to adapt it to them. This adaptation is performed by a transformation of the model that takes as input the initial design and the configuration of the feature model. This correspond to activity 4.

Feature modeling is one of the most extensively accepted techniques for modeling variability [15]. The specific model used in the present work follows the approach of Cardinality Based Feature Modeling, a widely used technique with proven usefulness in working with development frameworks [16].

To use a feature model as input or output for models transformations it needs to conform to a clearly defined structure or some sort of "metamodel". This structure must, however, be flexible enough to incorporate both the existing architectural and technological elements and any new ones that may arise in the future. For the model to have these features, we performed a study of some of the most used development frameworks (including Spring, Hiberate, Struts, JSF, CXF, Axis, DWR, etc.). More details on the analysis performed for the creation of the feature model and the decisions made for its creation may be found in [8].

At this point in the process, the architect must specify the design patterns and development frameworks on which to base the final design of the application, activity 5 in the diagram. To make this selection, the architect uses the information contained in the feature model, and then must link each element of the layer design to the architectural decisions that affect it, activity 6 in the diagram.

 ISBN: 978-1-61208-304-9

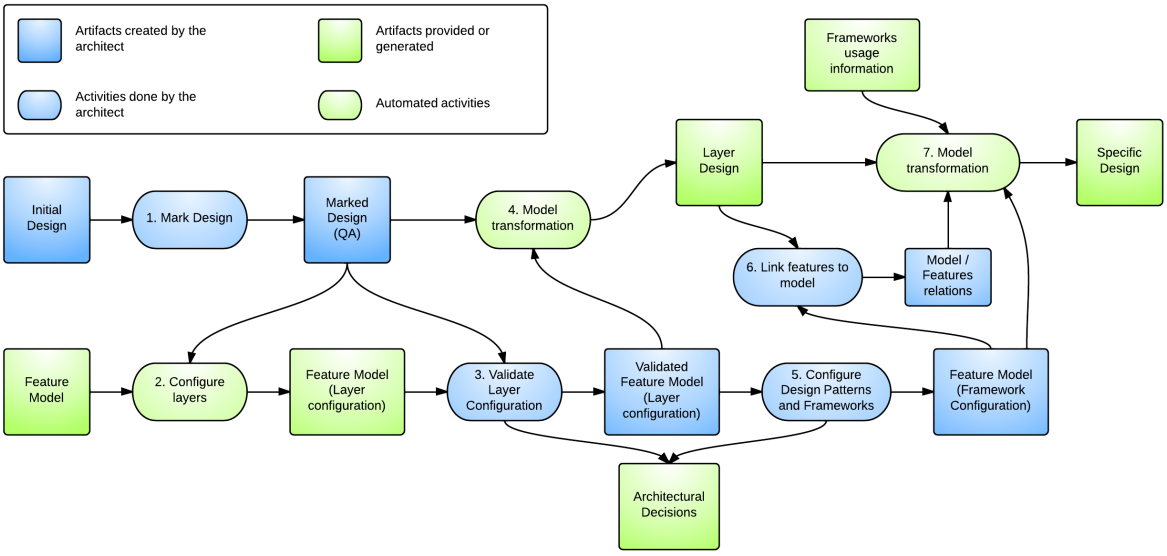

Fig. 2. The multi-layer application development process.

It should be noted that we propose a specific order for the decision making process, first the layers then the design patterns and finally the development frameworks. However, this order is not fixed and the architect can change it to suit their needs and preferences The abilities exhibited by features model to allow such flexibility were one of the main motivation to choose them as our architectural knowledge representation tool.

Finally, with all the information available, a model transformation is performed to convert the application layer design obtained previously into a specific design for the architectural decisions taken by the architect, activity 7 in the diagram. For this transformation, information is required about the development frameworks to be used. This information is included in the transformation by means of specific models describing the use of a particular technology.

The present work focuses on the architect's decision making. Specifically, in activities 3 and 5 in the diagram shown in Figure 2. To accomplish these activities, the architect uses three elements: the feature model containing information about the design patterns and the development frameworks that can be use for the development, the preliminary marked design that contains information about the relationship between the requirements and the system's quality attributes and his or her own knowledge about the system.

For a better understanding of this technique, we shall describe an example of how it works. Figure 3 shows the same activity diagram used previously enhanced with additional information about the quality attributes that the system must satisfy. Specifically, the verification of user permissions must meet security requirements, the notification of invalid access attempts should communicate with an external system and the modifications made by users should be auditable.

IV. Making Architectural Decisions

The elements presented in the previous section compose the basis for the architect's decision making. The present section will describe in detail the activities 3 and 5 in the diagram.

A. *Selecting layers*

A reasonable way to begin the decision making process when designing a multi-layer architecture is to choose the layers that will form part of the application. Many applications of this type use a common set of layers with similar functionality. Examples are the persistence, the presentation, and the Web service layers. The feature model we use contains a set of common layers, which can be easily expanded by adding new layers.

To simplify the architect's work, the information about the quality attributes added to the application's preliminary design can be used to offer an initial suggestion of an appropriate set of layers that might satisfy those attributes.

The layer suggestion process is based on a relatively simple set of rules. Specifically, a layer is suggested based on two criteria.

The first is the presence of certain elements in the preliminary design specific to each layer. The presence of these elements, which can be detected by querying the preliminary design model, determines whether a layer is to be proposed to the architect as part of the application's architecture. For example, the web services layer is suggested when the preliminary design includes interactions with external systems.

The second criterion is based on the marks with quality attribute information. Certain quality attributes entail the suggestion of certain layers. The presence of these marks is also detected by querying the design model. For example, whenever there appears an activity marked as Auditable the use of a log layer is suggested.

Table I shows a summary of the main criteria used to suggest the most common layers.

Technically these criteria consist of a set of model transformations that take as input the preliminary design and the

 ISBN: 978-1-61208-304-9

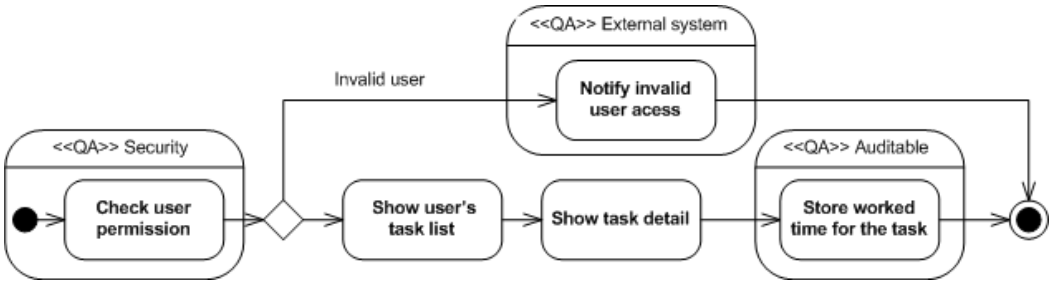

Fig. 3. Marked activity diagram.

TABLE I
LAYER SELECTION CRITERIA.

Layer	Criteria
Persistence	There is direct interaction with a Database or the same object is used in the activity diagram of more than one use case
Business logic	Always present, included here for further configuration at a lower abstraction level
Presentation	There is interaction with a human actor
Web services	There is interaction with external systems
Security	There is a Security mark on one or more of the elements in the UML diagrams
Log	There is a Maintainability mark on one or more of the elements in the UML diagrams
Test	There is a Testability mark on one or more of the elements in the UML diagrams

feature model. The output of this transformation is another model with an initial configuration of the feature model in which the suggested layers are selected.

Applying these criteria to the diagram shown in Figure 3, the architect is offered a basic initial selection of layers. This selection is presented in the form of a partial configuration of the feature model. In the case of the diagram in the figure, the architect will be proposed the use of the following layers: persistence because the activity diagram requires information to be retrieved that was stored in the system earlier and information to be stored for later use. Presentation because this layer includes all the elements related to interaction with the user. Web services because notifying an unauthorized access attempt requires communication with an external system. Security because checking the user's privileges has to be a secure task. And log because some of the diagram's activities have to be auditable.

An additional layer is suggested that encapsulates the application's business logic. This is a standard layer in enterprise applications to incorporate elements relating to the application's behaviour.

The architect's next task is to validate the set of suggested layers, or to modify it as may be deemed opportune. The final set of layers selected by the architect is registered as a partial configuration of the feature model and it is used to perform an initial model transformation. This transformation gives as output a specific design for the layers in which to split the application where each activity is represented in the layers in which it operates. Figure 4 shows a small fragment of the output of this transformation applied to a the preliminary design shown previously.

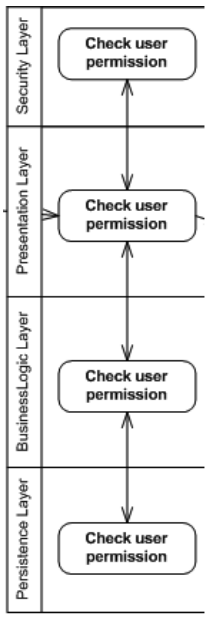

Fig. 4. Fragment of the layer adapted design.

B. Selecting patterns, technologies, and use

The following architectural decisions that have to be made consist in selecting the design patterns and technologies to use in the development of each of the layers identified in the previous section. It is possible, as was done during the selection of the layers, here too to present the architect with an initial selection based on the information contained in the initial design. Now, however, the architect's decisions have greater importance. In many situations, the choice of a particular technology will depend less on the application's requirements and more on either the criteria of the firm responsible for the development or the preferences of the architect. For example, the experience of the developers is one of the most important factors when selecting a technology to implement the MVC design pattern in a presentation layer. There are a number of frameworks that give full support to this pattern, the one which is normally used is that with which the architect or the development team has most experience. However, one should not forget that the chosen technologies must support the quality attributes of the application. So, the information about the quality attributes included in the preliminary design is most useful to validate the architect's decisions than to provide initial suggestions.

 ISBN: 978-1-61208-304-9

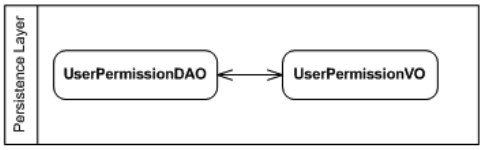

Fig. 5. Fragment of the design pattern adapted design.

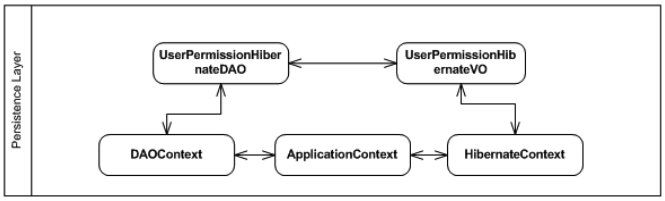

Fig. 6. Model transformation sequence.

Due to this, the weight of this task falls largely on the architect. It is generally done in two steps. In the first, the architect selects the design patterns to use in the development of each layer. To make this choice the architect uses the list of patterns available in the feature model for each layer and the preliminary design with the information about the functional and non-functional requirements of the application. Typically, the selection is that which can best fulfill the application's functional and non-functional requirements. However, the architect has the final say on the matter and can take architectural decision based on different criteria such us his or her own previous experience or the development team knowledge about specific technologies. In the example we are using, the architect could choose the MVC and Web Remoting patterns jointly for the presentation layer, and the ReST pattern instead of SOAP for the Web services layer. With this information, it is possible to apply a new partial transformation to obtain a more detailed design adapted to the design patterns chosen by the architect. Figure 5 shows a small fragment of the result of this transformation after applying the DAO pattern to an activity in the persistence layer.

In the second step, the architect must select which technology or development framework will be used to support each of the selected design patterns. Again, this set of architectural decisions is based on the information contained in the feature model and the preliminary design. The selection will be made from among the technologies specific to the design patterns chosen and will depend mainly on the architect experience. For example, in the case of ReST, the architect must choose a technology that will support it, omitting consideration of other Web service technologies. Also, the presence in the feature model of the constraints mentioned above prevents the architect selecting incompatible technologies, and provides suggestions of technologies that are closely related to those already chosen. With this information, it is possible to apply the last transformation to obtain design adapted to the architectural decisions. Figure 6 shows a small fragment of the result of this transformation after using the Hibernate framework to implement the DAO pattern.

As mentioned above, for clarity reason, in this paper we show how our proposal supports architectural decisions in a specific hierarchical order. However, the architect could choose a different order. The use of feature models to specify architectural knowledge and the architectural decisions make it possible whilst the consistence is kept during the transformation process.

Using the described process provides the architect with two major advantages. One is that the number of options to consider when making decisions is pruned, with irrelevant elements eliminated from the process, and allowing the architect to focus on the use of just an allowed set. The other advantage is that using the feature model provides a simple mechanism for storing architectural decisions. Every decision made by the architect is reflected as a configuration of the feature model, and these configurations are easily stored for reference and use in future developments. The firm's architects will thus have a set of design guidelines based on successes or failures in previous projects.

V. Support Tools

In order to validate the techniques proposed in this paper, a set of tools is under development targeted at providing support to the entire process described in Section 2.

For tasks related to the architectural decision making, the core of the present work, a feature model is used that is similar to that described in Section 3, and which contains information on more than a dozen of the commonest development technologies.

Regarding the architectural decisions themselves, the techniques described in this paper are supported by a custom-designed Eclipse plug-in for the creation of multi-layer architecture Java projects. To create one of these projects, the plug-in needs a feature model such as that mentioned above. The options that will be presented to the architect for decision making are obtained from this model. The plug-in configuration allows specification of the URL at which to search for the feature model. This permits a firm to have a centralized model, so that any updates to include new technologies or to remove any that have become outdated are immediately distributed to all its architects.

Once the feature model to be used has been obtained, the plug-in presents the architect with the decisions to be taken. The different decisions are presented to the architect as wizards pages. We opted for an interface of this kind to simplify the architect's task. While feature models are a widely used tool in the context of product lines, an architect specializing in multi-layer application development would not necessarily know this notation, so that its use would impose an additional burden.

VI. Related Work

In the area of architectural decision making, particularly stand out for their close relationship with our proposal two works of Zimmermann [10], [14]. They present a framework for the identification and modeling of recurring architectural decisions, and for converting those decisions into design guidelines for future development projects. In particular, Zimmerman proposes seven identification rules (IRs). These rules have their counterpart in our proposal. The main difference between our work and that of Zimmerman is the use made of

 ISBN: 978-1-61208-304-9

those architectural decisions. In his work, the main objective is to gather information for use in future projects. Our focus is on using that information to simplify the process of obtaining an specific design of the application on which architectural decisions are made.

In the field of Web application development, Melia & Gomez [17] propose an extension to the model-driven methods of Web application development. Their proposal is closely related to the present work. Both pursue the same goal – to increase the architect's reliability when using model-driven techniques to design the architecture of a Web application. Nevertheless, their work focuses on RIA development, while ours is intended to encompass the entire class of multi-layer applications. Also, unlike our proposal, that of Melia & Gomez does not allow for control of the technologies used to implement the application, and neither does it provide any mechanism to log and store the decisions made by the architect for later use.

Finally, the studies of Antkiewicz [16] and Heydarnoori et al. [18] are of particular interest in the area of framework-based software development. Antkiewicz's techniques allow the modeling of specific designs for certain frameworks, and these models are then used to generate the source code. Heydarnoori et al. propose a technique for automatically extracting templates for implementing concepts of development frameworks. Unlike our work, the proposed techniques are very code centric, and their creation requires expertise in each framework employed. Our work is aimed at increasing the level of abstraction in the sense of being able to start from a technology-independent design, and progress to obtaining the final specific design by using the decisions made by the architect and model transformations.

VII. Conclusions and Future Work

This paper has addressed the problems facing the software architect when designing a multi-layer architecture. The complexity of these architectures, the complex relationship between functional and non-functional requirements and the high number of development frameworks and how they affect the non-functional requirements complicate the architect's task. We have proposed a technique for simplifying the architectural decision making process by means of the use of a feature model and a marked preliminary design. The proposed technique forms part of a broader procedure to address the transition from an initial design of an application to a design adapted to the architecture and technologies selected by the architect. This is a complex process that requires deep technical knowledge of the technologies involved. This complexity can be significantly mitigated by using model-driven development processes.

The next steps related to the architect's decision making will be based on improving the feature model's constraints. They can be used to incorporate additional information about quality attributes of the technologies, such as the performance of a given framework or the level of integration of two technologies. This additional information could be used to provide the architect with fuller and more precise initial suggestions.

Acknowledgments

This work was partially funded by the Spanish Ministry of Science and Innovation under Project TIN2012-34945, as well as by the Autonomous Government of Extremadura and FEDER funds.

References

[1] M. Fowler, *Patterns of Enterprise Application Architecture*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[2] R. Johnson, "J2ee development frameworks," *Computer*, vol. 38, no. 1, pp. 107 – 110, jan. 2005.

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

[4] M. Raible, "Comparing jvm web frameworks," http://static.raibledesigns.com/repository/presentations/Comparing_JVM_Web_Frameworks_Jfokus2012.pdf, Jfokus, 2012.

[5] N. B. Harrison and P. Avgeriou, "How do architecture patterns and tactics interact? a model and annotation," *Journal of Systems and Software*, vol. 83, no. 10, pp. 1735–1758, 2010.

[6] L. Chung and J. C. S. do Prado Leite, "On non-functional requirements in software engineering," in *Conceptual Modeling: Foundations and Applications*, 2009, pp. 363–379.

[7] M. Dalgarno, "When good architecture goes bad," *Methods & Tools*, vol. 17, pp. 27–34, 2009.

[8] J. Garcia-Alonso, J. B. Olmeda, and J. M. Murillo, "Architectural variability management in multi-layer web applications through feature models," in *Proceedings of the 4th International Workshop on Feature-Oriented Software Development*, ser. FOSD '12. New York, NY, USA: ACM, 2012, pp. 29–36. [Online]. Available: http://doi.acm.org/10.1145/2377816.2377821

[9] J. Berrocal, J. García-Alonso, and J. M. Murillo, "Facilitating the selection of architectural patterns by means of a marked requirements model," in *ECSA*, ser. Lecture Notes in Computer Science, M. A. Babar and I. Gorton, Eds., vol. 6285. Springer, 2010, pp. 384–391.

[10] O. Zimmermann, "Architectural decisions as reusable design assets," *IEEE Software*, vol. 28, no. 1, pp. 64–69, 2011.

[11] R. Wojcik, F. Bachmann, L. Bass, P. Clements, P. Merson, R. Nord, and B. Wood, "Attribute-driven design (add), version 2.0," Software Engineering Institute, Tech. Rep. CMU/SEI-2006-TR-023, 2006.

[12] P. Avgeriou and U. Zdun, "Architectural patterns revisited - a pattern language," in *EuroPLoP*, A. Longshaw and U. Zdun, Eds. UVK - Universitaetsverlag Konstanz, 2005, pp. 431–470.

[13] K.-J. Stol, P. Avgeriou, and M. A. Babar, "Design and evaluation of a process for identifying architecture patterns in open source software," in *ECSA*, ser. Lecture Notes in Computer Science, I. Crnkovic, V. Gruhn, and M. Book, Eds., vol. 6903. Springer, 2011, pp. 147–163.

[14] O. Zimmermann, "Architectural decision identification in architectural patterns," in *WICSA/ECSA Companion Volume*, ser. ACM International Conference Proceeding Series, T. Männistö, M. A. Babar, C. E. Cuesta, and J. E. Savolainen, Eds., vol. 704. ACM, 2012, pp. 96–103.

[15] K. Czarnecki, S. Helsen, and U. W. Eisenecker, "Staged configuration through specialization and multilevel configuration of feature models," *Software Process: Improvement and Practice*, vol. 10, no. 2, pp. 143–169, 2005.

[16] M. Antkiewicz, K. Czarnecki, and M. Stephan, "Engineering of framework-specific modeling languages," *IEEE Trans. Software Eng.*, vol. 35, no. 6, pp. 795–824, 2009.

[17] S. Meliá, J. Gómez, S. Pérez, and O. Díaz, "Architectural and technological variability in rich internet applications," *IEEE Internet Computing*, vol. 14, no. 3, pp. 24–32, 2010.

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

 ISBN: 978-1-61208-304-9