Model Transformations for the Automatic Suggestion of 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 design of these architectures poses a significant challenge to the software architect, mostly due to the large number of design patterns and development frameworks that can be used in the development of these architectures. The present work proposes a set of model transformations to automatically suggest the design patterns and frameworks best suited to satisfy both the functional and non-functional requirements of the system. This technique is part of a broader procedure to facilitate the software architect's task of converting the preliminar design of an application into a specific design tailored to the software architecture.**

***Keywords*—Multi-layer architectures; design patterns; development frameworks; model transformation; architectural decisions.**

I. Introduction

The layer architectural pattern allows software architects to decompose a system into decoupled components called layers. Each layer provides services to the layer above and uses the services of the layer below. The use of this pattern benefits the modifiability, portability, and reusability of the final system [1]. Therefore, multi-layer architectures are those in which the system has been decomposed into two or more decoupled components in a vertical manner.

These architectures are one of the most common solutions to develop enterprise web applications, since they allow developers to focus on the application's business logic instead of its structural details. However, the responsibility for the effective use of these architectures lies with each individual development team [2]. Specifically, the figure of the software architect takes on particular importance since the architecture plays a very important role in the way the application will be developed [3].

Thus, a development success will largely depend on the architect's experience, expertise, and skill in avoiding the introduction of potential errors [4]. Defining the architecture requires the architect to follow an arduous and complex process for getting information on the system requirements and for making decisions about how to structure the application to comply with them [5]. First, the architect has to acquire a great knowledge on the requirements and the relationships between them [6]. Subsequently, the knowledge extracted from the analysis of the requirements is used as the basis for making decisions about how to structure the system [7]. This implies that the architect cannot make these decisions based on a single requirement; she must have a complete view of all the requirements and how they interact. This conjuncture complicates the architect's work and exposes her to situations in which a misinterpretation can lead to the selection of an incorrect architectural pattern.

This situation gets even more complicated due to the close relation between architectural patterns. The application of a given pattern favors the selection of other patterns [8]. Therefore, the incorrect selection of a pattern can lead the architect to make incorrect decisions during the refinement of the architecture. This may cause the final design to fail the requirements of the system, jeopardizing the success of the project. Development frameworks, one of the most used tools in complex software development [9], complicate this problem. The increasing amount of frameworks and their rapid evolution rate [10] make it really difficult to keep up-to-date knowledge about them.

In this paper, a set of model transformations is presented to automatically suggest the architectural decisions best suited for each project. The transformations take as input the initial design of a system, including both functional and non functional requirements, and provides a set of architectural decisions, including design patterns to be applied and development frameworks to be used in the development, that would help the system meet its requirements. This work forms part of a broader proposal that covers the entire process of designing multi-layer applications.

The rest of this communication is organized as follows. Section 2 motivates this work by introducing the process of which the presented transformations are part of. Section 3 details the proposed transformation for automatically suggest architectural decisions. Section 4 specifies the validation performed over the transformation. Section 5 gives a review of the most significant related work. Finally, Section 6 presents the conclusions to be drawn from this work, and some indications of future work planned in this line of research.

II. Multi-layer Enterprise Applications

Figure 1 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

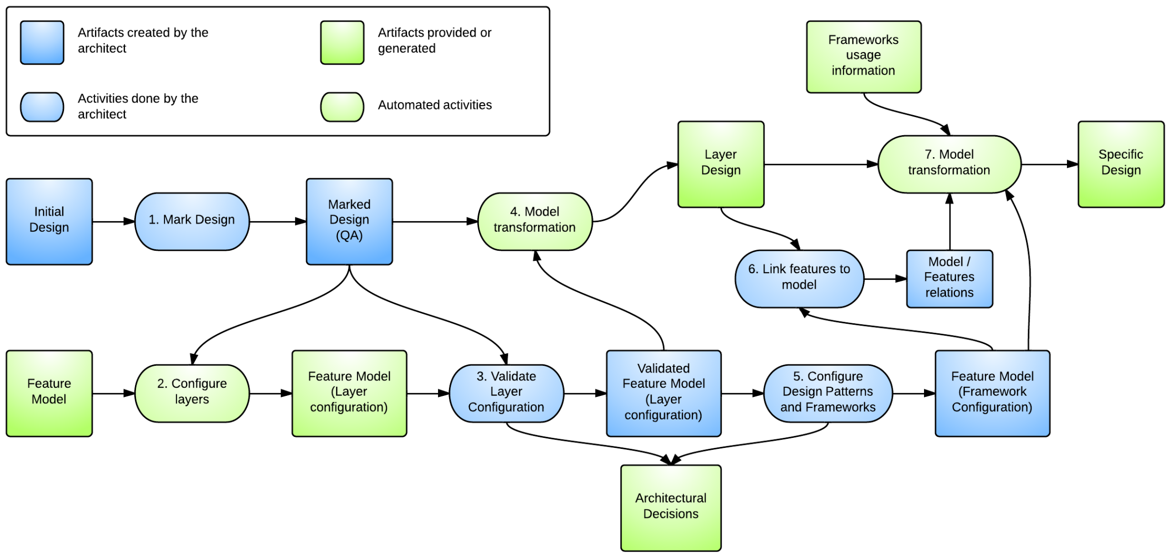

Artifacts created by the architect
Artifacts provided or generated
Activities done by the architect
Automated activities
Initial Design
1. Mark Design
Marked Design (QA)
4. Model transformation
Layer Design
Frameworks usage information
7. Model transformation
Specific Design
6. Link features to model
Model / Features relations
Feature Model
2. Configure layers
Feature Model (Layer configuration)
3. Validate Layer Configuration
Validated Feature Model (Layer configuration)
5. Configure Design Patterns and Frameworks
Feature Model (Framework Configuration)
Architectural Decisions

Figure 1. The multi-layer application development process.

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.

Usually, the relationship between functional and non-functional requirements are not explicitly detailed [11]. 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 Berrocal et al. [12].

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, more details on the decision-making process followed by architects may be found in [13].

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 [14]. 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 [15].

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 [16].

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.

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 model transformations used to offer a set of viable architectural suggestion to the architect; specifically, on activities 2 and 5 in the diagram shown in Figure 1. To accomplish these activities, the transformation use two elements: the feature model containing information about the design patterns and development frameworks that can be used for the development and the preliminary marked

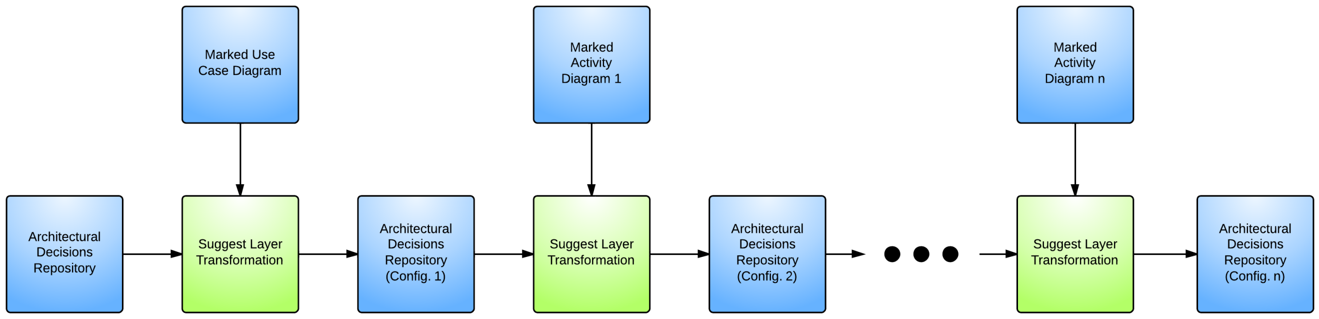

Marked Use Case Diagram
Marked Activity Diagram 1
Marked Activity Diagram n
Architectural Decisions Repository
Suggest Layer Transformation
Architectural Decisions Repository (Config. 1)
Suggest Layer Transformation
Architectural Decisions Repository (Config. 2)
Suggest Layer Transformation
Architectural Decisions Repository (Config. n)

Figure 2. Layer Suggestion Transformation application diagram

design that contains information about the relationship between the requirements and the system's quality attributes.

III. AUTOMATIC SUGGESTION OF ARCHITECTURAL DECISIONS

This section will describe in detail the model transformation used in activities 2 and 5 of the process presented above to automatically provide a set of architectural decisions that can be used for the architect to design a multi-layer architecture that meets the requirements of the system.

A. *Automatic layer suggestion*

The first model transformation presented is the *Layer Suggestion Transformation*. Figure 2 shows the elements of the process involved in the application of this transformation.

The goal of this transformation is to provide to the architect a possible set of layers to be used in the development of a system. For this, the transformation take as input a feature model containing the set of possible architectural decisions and the initial design of the system marked with information about the QAs it must fulfill. With this information, the transformation generates a copy of the feature model in which the suggested layers has been selected.

As shown in Figure 2, the transformation is designed in such way that it can be applied multiple times, if the initial design of the system is described in several models. Each application of the transformation generates an enriched layer suggestion that can be used as the input of the next application of the transformation. The final result obtained is the set of layers suggested to implement all the elements contained in the different initial design models.

The transformation will suggest a given layer based on specific features found in the initial design of the application or based on the marks containing information about the QAs of the system. Figure 3 shows a fragment of the transformation that suggest the use of a persistence layer if the initial design model contains any DataStore elements.

Figure 5 shows a fragment of the transformation that suggest the use of a security layer if any element of the initial design model is annotated with the given QAs.

This simple set of criteria for layers suggestion can be adapted to meet company policies or architects preferences by

```
if(f.isPersistenceConfiguration()){
    if(UML!DataStoreNode.
        allInstances().size()>0){
        t.state <-#USER_SELECTED;
    }
}
```

Figure 3. Persistence layer suggestion transformation

```
for (stereotype in UCP!Stereotype.
    allInstances()){
    if(stereotype.name='Authenticity
        ' or stereotype.name='
        Security' or stereotype.name=
        'Confidentiality'){
        for (useCase in UML!UseCase.
            allInstances()){
            if(useCase.isAnnotated(
                stereotype)){
                t.state <-#
                    USER_SELECTED;
            }
        }
    }
}
for (stereotype in AP!Stereotype.
    allInstances()){
    if(stereotype.name='Authenticity
        ' or stereotype.name='
        Security' or stereotype.name=
        'Confidentiality'){
        for (activityPartition in
            UML!ActivityPartition.
            allInstances()){
            if(activityPartition.
                isAnnotated(
                stereotype)){
                t.state <-#
                    USER_SELECTED;
            }
        }
    }
}
```

Figure 5. Security layer suggestion transformation

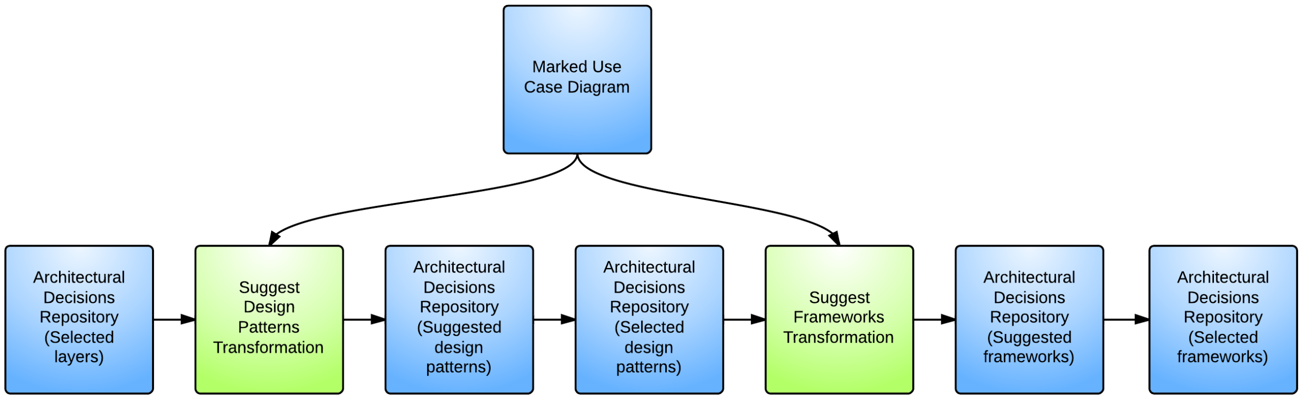

Marked Use Case Diagram
Architectural Decisions Repository (Selected layers)
Suggest Design Patterns Transformation
Architectural Decisions Repository (Suggested design patterns)
Architectural Decisions Repository (Selected design patterns)
Suggest Frameworks Transformation
Architectural Decisions Repository (Suggested frameworks)
Architectural Decisions Repository (Selected frameworks)

Figure 4. Design Patterns and Frameworks Suggestion Transformation application diagram

```
if(thisModule.halfUCAnnotated(
    stereotype)){
     t.state <-#USER_SELECTED;
}

helper def : halfUCAnnotated(s : UCP
    !Stereotype): Boolean =
     if (UML!UseCase.allInstances()->
         collect(    ext | ext.
         extensionPoint).flatten()->
         select(exten | not exten.
         getAppliedStereotype(s.
         qualifiedName).oclIsUndefined
         ()).size()/ UML!UseCase.
         allInstances().size() >= 0.5
             then
          true
     else
          false
     endif;
```

Figure 6. Alternative security layer suggestion transformation

enriching the transformations that suggest each of the layers. Figure 6 shows an alternative to the security layer suggestion that only select such layer if half or more of the use cases of the initial design are marked with the given QAs.

The final product obtained by this transformation is a configuration of the feature model in which the suggested layers are selected. This model will be later used by other transformations to further advance in the development process and can also be used or modified by the software architect.

B. Automatic design patterns and frameworks suggestion

The next model transformation presented is the *Design Patterns and Frameworks Suggestion Transformation*. Figure 4 shows the elements of the process involved in the application of this transformation.

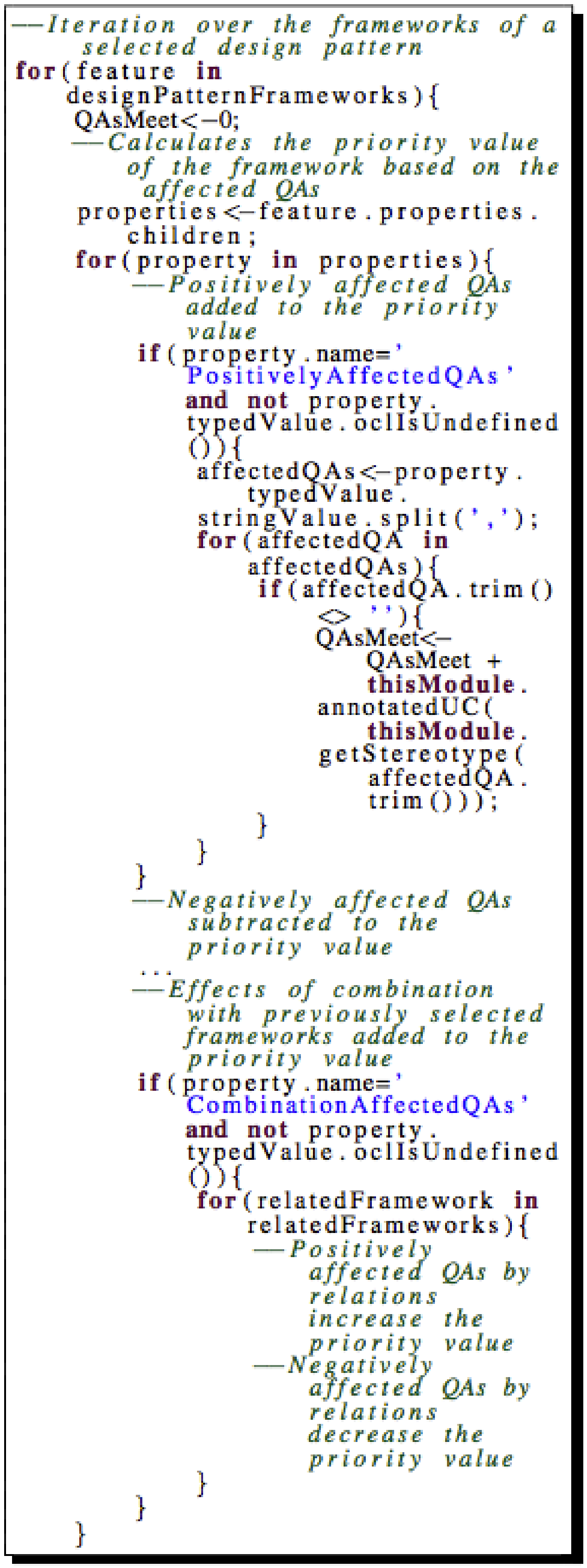
The goal of this transformation is to provide to the architect a possible set of design patterns and frameworks to be used in the development of a system. To do this, the transformation is divided in two. The first one take as input the set of layers selected and the marked use case diagram. With this information, the transformation generates a copy of the feature model in which the suggested design patterns have been selected. The second transformation take as input the previously generated set of selected design patterns and the marked use case diagram and generates a copy of the feature model in which the suggested frameworks have been selected.

The transformation has been divided in two steps in order to give architects the opportunity to refine or validate each level of suggestion independently. Thus, the set of selected design patterns using in the second part of the transformation are not necessarily the ones automatically suggested by the transformation but the ones validated by the architect.

To suggest a particular design pattern or framework the transformation uses the information about the QAs affected by them included in the feature model. This information is checked against the QAs the system must fulfill, as indicated by the marks included in the use case diagram, not forgetting the effect the combination of different design patterns and frameworks has on the final system QAs. Figure 7 shows a fragment of the algorithm used to suggest a framework on the basis of such information.

For each selected design pattern this prioritization algorithm suggest the framework that best helps to fulfill the system QAs based on the framework influence in the QAs and in the relations with the already selected frameworks. The final product obtained by this transformation is a configuration of the feature model in which the suggested design patterns and frameworks are selected. This model will be later used by the last transformation to further advance in the development process and can also be used or modified by the software architect.

The transformation fragment showed in Figure 7 calculates the priority value of a specific framework given the positively and negatively affected QAs by such framework and by its combination with the rest of the frameworks already selected. The framework with the highest priority value is suggested to be used in the development



```
--Iteration over the frameworks of a
    selected design pattern
for(feature in
      designPatternFrameworks){
    QAsMeet<-0;
    --Calculates the priority value
        of the framework based on the
        affected QAs
    properties<-feature.properties.
        children;
    for(property in properties){
        --Positively affected QAs
            added to the priority
            value
        if(property.name='
            PositivelyAffectedQAs'
            and not property.
            typedValue.oclIsUndefined
            ()){
            affectedQAs<-property.
                typedValue.
            stringValue.split(',');
            for(affectedQA in
                affectedQAs){
                if(affectedQA.trim()
                    <> ''){
                    QAsMeet<-
                        QAsMeet +
                        thisModule.
                    annotatedUC(
                        thisModule.
                    getStereotype(
                        affectedQA.
                        trim()));
                }
            }
        }
        --Negatively affected QAs
            subtracted to the
            priority value
        ...
        --Effects of combination
            with previously selected
            frameworks added to the
            priority value
        if(property.name='
            CombinationAffectedQAs'
            and not property.
            typedValue.oclIsUndefined
            ()){
            for(relatedFramework in
                relatedFrameworks){
                --Positively
                    affected QAs by
                    relations
                    increase the
                    priority value
                --Negatively
                    affected QAs by
                    relations
                    decrease the
                    priority value
            }
        }
    }
```

Figure 7. Framework suggestion transformation

IV. VALIDATION

This section tries to detail the usefulness of the presented transformations. To validate them, they have been applied to two industrial project. Industrial projects were used instead of other validation methods since, to properly ensure their impact and benefits, reasonably large projects were needed.

The two projects involved the development of a medium-size multi-layer application. In each project, the transformations presented here were used and the following features were evaluated: their feasibility, their completeness and the effort required to apply them.

The results obtained evaluating the feasibility of the transformations were very positive. All the information available in the process was used to suggest a set of architectural decision by evaluating every architectural decision posible and choosing the best suited to the system requirements. The only feasibility drawbacks were found on the usability of the transformations. Some of the detected problems were fixed, but additional effort is needed in that regard. In general, the performed validation strongly support that the transformations are feasible, i.e., they can be applied to real-life examples by averagely trained personnel.

The results obtained assessing the completeness of the transformations were encouraging. A significant amount of the architectural decisions taken during the projects were automatically suggested by the transformations. The goal of the process, of which the presented transformations are part of, never was to include the complete range of development frameworks, but to provide a mechanism flexible enough to admit all of them. However, this flexibility has some disadvantages, namely the transformations will never be complete because there will always be a new technology to add. Summarizing, the data collected support that the transformations are complete. They facilitate the use of a broad range of architectural decisions and development frameworks, which is very useful for the development multi-layer applications but they have to be constantly updated to keep up with technological evolution.

The results obtained assessing the effort required to use the transformations are very promising. The use of the transformations causes a small overhead in effort needed, but its effect is diluted in the time saved during development. Additional effort are needed when new architectural decision or technologies have to be included into the transformations. This additional effort can be a major drawback using them. Their potential benefits are shown more clearly where no additional element has to be taken into account. Concluding, the data collected strongly support the effort needed characteristic, indicating that the use of the proposed transformation reduces the total effort spent in the design and development of multi-layer applications.

V. RELATED WORK

In the area of architectural decision making, particularly stand out for their close relationship with our proposal two works of Zimmermann [17][18]. 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 transformations. The main difference between our work and that of Zimmerman is the use made of those architectural decisions. In his work, the main objective is to gather information for use in future projects. Our focus is on automatically suggesting the best suited decisions to meet the requirements of the system being developed.

In the field of Web application development, Melia et al. [19] 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, the approach in [19] 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 et al. [15] and Heydarnoori et al. [20] 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.

VI. 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, a marked preliminary design and a set of model transformations to automatically suggest the best suited design patterns and development frameworks. 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 and the model transformations presented in this work will be based on improving the prioritization algorithm used to suggest the most appropriate development framework based on the QAs affected by it. This algorithm can be improved by taking into account the frameworks that has not been selected but that can improve the system QAs if they are chosen over the architect manual selection.

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] P. Avgeriou and U. Zdun, "Architectural patterns revisited - a pattern language," in EuroPLoP, 2005, pp. 431–470.

[2] R. S. Pressman, "Manager - What a tangled web we weave," IEEE Software, vol. 17, no. 1, 2000.

[3] L. Northrop, "The importance of software architecture," http://sunset.usc.edu/GSAW/gsaw2003/s13/northrop.pdf, SEI, Carnegie Mellon University, [retrieved: 07, 2014], 2003.

[4] M. Dalgarno, "When good architecture goes bad," Methods & Tools, vol. 17, no. 1, 2009, pp. 27–34.

[5] P. Clements, R. Kazman, M. Klein, D. Devesh, S. Reddy, and P. Verma, "The duties, skills, and knowledge of software architects," in Proceedings of the Sixth Working IEEE/IFIP Conference on Software Architecture, ser. WICSA '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 44–47.

[6] R. Capilla, M. A. Babar, and O. Pastor, "Quality requirements engineering for systems and software architecting: methods, approaches, and tools," Requir. Eng., vol. 17, no. 4, 2012, pp. 255–258.

[7] P. C. Clements, "On the importance of product line scope," in PFE, 2001, pp. 70–78.

[8] 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, 2010, pp. 1735–1758.

[9] I. Vosloo and D. G. Kourie, "Server-centric web frameworks: An overview," ACM Comput. Surv., vol. 40, no. 2, 2008.

[10] M. Raible, "Comparing JVM web frameworks," Jfokus, [retrieved: 07, 2014], 2012. [Online]. Available: http://static.raibledesigns.com/repository/presentations/Comparing_JVM_Web_Frameworks_Jfokus2012.pdf

[11] 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.

[12] 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.

[13] J. Garcia-Alonso, J. B. Olmeda, and J. M. Murillo, "Architectural decisions in the development of multi-layer applications," in Proceedings of the 8th International Conference on Software Engineering Advances, ser. ICSEA '13, 2013, pp. 214–219.

[14] 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, 2005.

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

[16] 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.

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

[18] ——, "Architectural decision identification in architectural patterns," in WICSA/ECSA Companion Volume, 2012, pp. 96–103.

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

[20] A. Heydarnoori, K. Czarnecki, W. Binder, and T. T. Bartolomei, "Two studies of framework-usage templates extracted from dynamic traces," IEEE Trans. Software Eng., vol. 38, no. 6, 2012, pp. 1464–1487.