Towards a Dynamic Model for Requirements Handling in Software Architecture Configuration Management

Razie Alidoosti, Shahrouz Moaven, Jafar Habibi, and Mohammad Mehdi Khalili

Abstract—Nowadays the role of software architecture is obvious. Both complex requirements and non-deterministic changing environments lead software architecture to face with problems for coverage of requirements and environments. Consequently, the configuration of the software architecture is faced with challenges even in run-time. Being able to deal with such problems requires new techniques to be used. In spite of variety of related works performed in this domain, improvements are required in some aspects such as consideration of overlap and contradictory requirements and also effects in dynamic environments. In this paper, a dynamic model is presented for handling changing requirements which makes use of the configuration management concepts such as versioning, composition, consistency, construction, and selection; hence the possibility of configuring software architecture in overlapped and contradictory requirements and dynamic environments has been provided in the proposed model. The model includes change management and versioning activities for software architecture and its artefacts. The results of performing the proposed model on a case are reported in this paper.

Index Terms—Dynamic model, software architecture, requirements handling, configuration management.

I. INTRODUCTION

Today the need for interoperability of software systems is of importance more than ever. Therefore satisfaction of special requirements of this domain requires solutions that include identifying and handling quality attributes such as availability, flexibility, etc. One of the most important requirements is the run-time configuration management considered as the most prominent requirement of dynamic environments. Adaptable software architecture is able to be adjusted in variable environmental conditions [1].

Changes can occur at different times, such as: design time, before implementation, or at run-time [2]. Various factors lead to changes in the software architecture, e.g. changing customer needs, need to create new software faster, adding new software features and modifying soft-ware faults during maintenance phase [3]-[5]. That order to fix these changes making use of configuration management and its concepts.

The architecture has been integrated with the requirements in such a way that it is impossible to judge which of them came earlier than other [4]. Although there is a relationship between architecture and software requirements, but there are different opinions on the nature of this relationship [3]. In fact, requirements are description of problem but software architecture represents the structure of software system that solves the problem with its main elements such as components and connectors. In fact, because of the different nature of the two, there is a gap between them.

Despite much work performed in identifying and meeting requirements in the area of configuration management of software architecture, there is still no perfect proven solution. This problem is obvious further in run-time where we need quality attributes such as availability, performance, robustness, reliability, etc [6].

Among the work done, [7] can be pointed which presents a method for specifying and executing dynamically adaptive software systems in which model-driven and aspect-oriented techniques are combined. Also the model-driven approach presented in [8] provides multiple architectural run-time models at different levels of abstraction as a basis for adaptation. This approach takes advantage of run-time models and model-driven engineering techniques to adapt. A model for reconfiguration coordination patterns was introduced in [9]. This approach shows how reconfiguration management patterns are stated by combination of elementary transformations in model.

The interaction between software requirements and architecture is one of the most challenging problems in software engineering researches. To address this challenge, we will examine the relationship between these two.

Without any doubt one of the most important issues during the evolution of the system or the long-term maintenance is considerations and requirements contained in the existing requirements in the field of quality attributes of the system. The main reference accountable to quality attributes is software architecture. Hence changes in the architecture and performing operations such as versioning, composition, consistency, etc are unavoidable in this circumstance.

We need to perform existing issues such as versioning, change management, etc. for configuration items (CIs) of software architecture correctly and accurately. In this article we take advantage of a run time model to handle requirements on the software architecture using the concepts of software configuration management (SCM) such as change control, versioning, composition, consistency, construction and selection.

Another objective of this paper is to emphasize the importance of changing and improving the adaptability of the architecture. Performing configuration management in architecture level and applying its changes cause

Manuscript received November 10, 2014; revised January 7, 2015.
R. Alidoosti, S. Moaven, and J. Habibi are with the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran (e-mail: alidoosti@ce.sharif.edu, moaven@ce.sharif.edu, jhabibi@sharif.edu)
M. M. Khalili is with the Department of Computer Engineering and Information Technology, Amirkabir University of Technology, Tehran, Iran (e-mail: mmkhalili@aut.ac.ir).

DOI: 10.7763/LNSE.2016.V4.240
more understandable and exact configuration in lower abstraction levels such as code and design.

The remainder of this paper is as follows. First definitions about the software architecture and software configuration management concepts and requirements are outlined. Then a summary of some of approaches and models used for architectural adaptability is described. In the subsequent section, dynamic model for requirement handling is presented that provides a process for control of architectural requirements. Next section presents the results in a case study follows by conclusion and future work in two last sections.

II. BASIC CONCEPTS

A. Software Architectural Requirements

The software architecture is a critical aspect in software systems which should be considered by architects for describing high level structure of systems. Software architecture should be designed to meet the major functional and non-functional requirements.

As systems have become complex over time, design of their architecture is complicated and obtaining desired requirements is difficult [10]. Software architecture is known as a high level structure of software which is de-fined as set of components, connectors, their properties, and constraints and also illustrates various aspects of system [11].

Software architecture explores variety of existing relations in software engineering such as relationship between architecture and related quality attributes and is considered as one of the existing ways to facilitate achievement of functional and non-functional requirements [12]. System requirements describe features and system characteristics such as functional and non-functional. Requirements can be simple or complex, explicit or implicit, brief or detailed. Requirements can be considered as key inputs for the design of software architecture. In fact, requirements and quality attributes are two most important issues during the evolution of the system which software architecture is responsible to satisfy them. In fact, requirements play an important role in the success of software systems and architectures associated with them [13].

B. Configuration Management

Software configuration management is for control of evolution of software systems. On the other hands, configuration management is a discipline for recorded information about configuration items [11]. Configuration management is a task embedded in any system for change control and its main usage is in recent complex and big systems [12].

Architectural configuration management is used to keep integration of elements in any software projects. Since growth and changes are inevitable in software systems, configuration management is considered as an integral element of software development and maintenance [14]. Software configuration management consists of activities such as: change management, version management, build system, and release management.

Issues of both software architecture and configuration management are associated with each other and the relationship becomes more highlighted especially when changes in the architecture are carried out. In fact with the change in the system and ultimately change in the architecture, the architectural reconfiguration management is essential.

III. RELATED WORK

In this section, the related research performed in software architecture and software configuration management domains to handle and control requirements are represented. In review of the works related to adaptation we take advantage of different techniques such as architecture-based techniques, component-based techniques, and techniques and methodologies to dynamic adaptation.

In [15] a comprehensive adaptation technique has presented that is able to adapt to different requirements. Two managers are used in this technique; one for the adaptation and another for evolution. The model-driven approach presented in [16] provides multiple run-time architectural models at different levels of abstraction as a basis for adaptation. In [17] a new method called Molhado is presented to manage the evolution of software systems at a logical level. Molhado Architectural Software Configuration Management System is a system model in which architectural concepts are integrated with system model, which enables management of non-planned evolution of the software architecture. In this model, the configurations are kept consistent not only with the source code but also with high-level soft-ware architecture. The model generalized for different architecture description languages and various architectural styles.

In [18] a method have been proposed for specifying and executing adaptive dynamically in which software systems combined model-driven and aspect-oriented techniques to help engineers to control complexity of such systems, while offer providing a higher degree of automation and evaluation.

In [19] a framework called RESAS has been proposed to implement real-time software systems; it allows real-time systems to be adapted with time constraints and hardware failures. Adaptive software architecture for digital signal processing applications has been presented in [20]. This architecture is a signal flow graph, which allow to dynamic change the graph nodes for change order and run sequence on the fly. The model introduced in [8] is based on a collection of primitives to reconfigure the connector as well as a language to specify the interfaces and reconfigure them. A system model for managing the evolution of software architecture, called Ma, was introduced in [9] in which users specify architectures in a traditional way. Also this model com-bines concepts of configuration management with technicals of software architecture field. A range of styles was examined in [7] and evaluates them according to the four elements of a assess framework called BASE. This framework was intended
how a style support changes for behavior, mood, implementation and support of asynchrony change. Styles used in this context include: REST, event-based, service-oriented, and peer-to-peer. Architectural models used for describing the dynamic variability of applications were presented in [13].

Also considerable research performed in [21]-[23] in reconfigurable dynamic environments which provided for dynamic reconfigurable models, protocols, and change management techniques and description and design software architecture.

Category 1 (C-1), includes techniques, approaches, and methods provided for adaptation and evolution of software systems. Category 2 (C-2) includes frameworks, architectures, and models and Category 3 (C-3) represents environments in these fields.

TABLE I: KPI FOR CATEGORIES (CK)

<table>
<thead>
<tr>
<th>Quality Indicator</th>
<th>C-1</th>
<th>C-2</th>
<th>C-3</th>
<th>Our Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run-time support</td>
<td>0.441</td>
<td>0.343</td>
<td>0.439</td>
<td>0.484</td>
</tr>
<tr>
<td>Flexibility</td>
<td>0.352</td>
<td>0.434</td>
<td>0.272</td>
<td>0.566</td>
</tr>
<tr>
<td>Variability</td>
<td>0.522</td>
<td>0.250</td>
<td>0.400</td>
<td>0.411</td>
</tr>
<tr>
<td>Completeness</td>
<td>0.537</td>
<td>0.496</td>
<td>0.496</td>
<td>0.666</td>
</tr>
<tr>
<td>Tool Support</td>
<td>0.640</td>
<td>0.375</td>
<td>0.401</td>
<td>0.555</td>
</tr>
</tbody>
</table>

The KC (KPI for Categories) supports categories candidate for different indicators and the CK (Categories in KPI) ranks which candidate categories are best at each quality attribute. These two tables were achieved from an experimental study by the presence of seven domain experts with more than ten years of experience in this field [13].

TABLE II: CATEGORIES IN KPI (CK)

<table>
<thead>
<tr>
<th>Quality Indicator</th>
<th>C-1</th>
<th>C-2</th>
<th>C-3</th>
<th>Our Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run-time support</td>
<td>0.711</td>
<td>0.706</td>
<td>0.550</td>
<td>0.729</td>
</tr>
<tr>
<td>Flexibility</td>
<td>0.765</td>
<td>0.543</td>
<td>0.388</td>
<td>0.649</td>
</tr>
<tr>
<td>Variability</td>
<td>0.695</td>
<td>0.555</td>
<td>0.333</td>
<td>0.588</td>
</tr>
<tr>
<td>Completeness</td>
<td>0.617</td>
<td>0.765</td>
<td>0.500</td>
<td>0.649</td>
</tr>
<tr>
<td>Tool Support</td>
<td>0.473</td>
<td>0.578</td>
<td>0.732</td>
<td>0.416</td>
</tr>
</tbody>
</table>

Table I was constructed from pairwise comparison of categories candidate for all quality attributes and table 2 is for each quality attribute for all candidate categories. Coloring cells of table indicates how the comparison has been done.

Run-time of our approach is more than all C-1 and C-2. Also flexibility in our approach is better than all categories at table 1 but at Table II is only more than C-2 and C-3. As a result we can conclude that this indicator is better than C-2 and C-3; however we can’t conclude about C-1.

TABLE III: SCALE FOR PAIRWISE COMPARISON USING AHP [13]

<table>
<thead>
<tr>
<th>Relative Intensity</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Of equal importance</td>
</tr>
<tr>
<td>3</td>
<td>Slightly more important</td>
</tr>
<tr>
<td>5</td>
<td>Highly more important</td>
</tr>
<tr>
<td>7</td>
<td>Very highly more important</td>
</tr>
<tr>
<td>9</td>
<td>Extremely more important</td>
</tr>
</tbody>
</table>

IV. DYNAMIC MODEL FOR REQUIREMENT HANDLING

First, we have identified a set of requirements associated with the architecture, based on architectural angles of view, such as response time, security, availability, multi-user system and also consider evaluation criteria by a group of experts. Each of these can be effect on different architectural phases such as analysis, design, etc.

Then selects cases associated with configuration of software architecture from among selected requirements, according to indicators. Next choose the proper set from among large number of configuration items. These items can be selected in accordance with such criteria as security criticality, human safety criticality, mission criticality, financial criticality, design newness, interface criticality, hardware integrity, status sensitivity, technology sensitivity, etc.

These criteria also have different degrees of importance. Generally, experts will comment on each of the items. And the end result will be taken by consensus and
their colleagues.

One of main goals of this model facilitates the selection of configuration items, Fig. 1.

Fig. 2 is presented a generic architecture for the dynamic model. The four-layer of this architecture are horizontal and its two layers are vertical.

Knowledge management vertical layer, in addition to basic knowledge, keep knowledge about the various layers and components of the model considering different the changes and versions. This layer and also versioning are as cross cut and impact on all horizontal layers.

Horizontal layers transmitted data with each other. In the lowest layer, first receive requirements and then based on knowledge screening.

In the second layer, using a mechanism of decision and also consider changes to choose proper items. Recorded the current situation in this part is to improve the quality of decisions in this layer.

In the third layer, using existing knowledge in the knowledge base for applies the changes on the architectural and their communication and eventually results will be saved in the repository. Finally, in the top layer is done the testing and evaluation of activities.

Changes are done on each component of the architecture but slight change alone not changes the version of architecture. But slight changes on the architectural components and connectors integrate and according architectural baselines, to be recorded change of a version.

Fig. 2. The proposed architecture for dynamic model.

For example, as shown in Fig. 3 in baseline X2 and after integrate the changes, components and connectors transferred from version B to version C. Identify baseline in proposed model is different for every system and is done by mediator component. Time to change one version to another is important. Hence, in case of a problem allows possibility return to a previous version is only base on existents baselines.

![Versioning component in proposed model.](image)

To solve this problem, we use a type of change management and versioning on software architectural components in our proposed model. This makes it possible for us to return step by step to the previous version of the architectural components and connectors in case of problem.

However, it is possible to guarantee the accuracy and consistency of changes in change management component with respect to model and the hierarchical structure and also the decision-maker component. It should be noted that the fine grained levels of activities are performed in architectural components level and do this activity will not damage the time and cost of performing these activities. As specified in model, we act independently for slight changes, Fig. 4.

![Versioning component in proposed model.](image)

In the proposed model, α component is embedded for detecting the effect of change or required requirements of the system and its impact on software architecture. This component is based on its inputs and utilizes the existing rules in its knowledge management repository to identify type of changes such as slight change or general change. Then on this basis, changes according to its fine grained level will be placed on one of the two directions P or Q. Slight changes will be sent on P direction and will be applied on CIs in β component; changes will be integrated (ΣC) and returned to α component next. By its own diagnosis, α component will send these changes or other changes to Q direction; changes will be applied on CIs in γ component. It should be noted that all these actions will be recorded in the knowledge base.

After applying various changes on the CIs, the items will be sent to the δ component for testing. There exist test types such as unit testing, integration testing, system testing, and acceptance testing is conducted on them and if, for example perform unit test on CIs in P direction and integration test on CIs in Q direction. In case of existence of a problem, previous components will be used again.

Decision making is one of the main activities undertaken in this component. In this case, first changes are done in which of two directions and second, that this changes is effect on which of the CIs. As shown in the figure, is considered four categories of changes such as Requirements and Development
Errors (R&D E) is a type of discrepancy in requirements and development. Violations of Standards (VoS) are another type of discrepancy that violates development standards. Unimplementable Requirements (UR), Enhancements and Improvements (E&I) are change requests.

V. CASE STUDY

To clarify the way the model works, we use a case that has been used in different studies. By using this case feasibility of the proposed can be examined.

We will perform the proposed dynamic model on KLAX video game application. For this application C2-style architecture has been used.

In order to display run-time configure and respond to requirements in a better way, we used from a high score list to KLAX [24] that includes three components: High Score ADT, High Score Logic, and High Score Artist. High Score ADT component maintains name and score of the first ten players. High Score Logic component makes decision about when the need to change high score list. High Score Artist component is used to display high score list by providing a user interface.

In fact, for demonstrating the performance and functionality of our model we use three well-known scenarios which will be discussed below.

In the first scenario, we place letters rather than tiles. In this scenario, the requirements will be classified after entering in the model and decision maker component will decide about type of changes according to C2 style and its flexibility. This component specifies whether this change is a major one to the architecture or not. Also it detects whether a change for a component or connector is internal or not. These components have been specified in layers 2 and 5 by blue.

After placement of letters, they will be unit tested. If layers were selected correctly, so no errors but when bellow component not select, it must be selected in the integration test in Fig. 5.

In the second scenario, we want to add high score list to our model. The component specify in layers 1, 2 as red color. After inter requirements and they integration together, the expert will know in the current state, cannot handle it with change components on the system. In other words, the requirement is the need to basic change, but given the flexibility and extensibility of C2 style, the architecture will not hurt. After adding these three components we will be able to perform unit testing and integration, Fig. 6.

In the third scenario, the two components which was added in scenario 2, will be changed. These components are selected and integration test will be performed on them. These components are specified in layers 1 and 4 as purple color, Fig. 7.

The model handles and change behavior dynamically based on input that an expert imports in the field. It should be noted that the experts have used practical application. The results can be seen.

VI. CONCLUSION AND FUTURE WORK

Software architecture has the potential to provide a foundation for run-time modification, but
since have complex requirements and non-deterministic changing environments; face with challenges in its configuration and also cover requirements in maintenance and run time [25, 26]. We reviewed variety of performed works and existing challenges in this field; however they demonstrated that a complete solution has not been provided yet. Hence in this paper a dynamic model has been presented that takes advantage of existing concepts and mechanisms in configuration management field such as versioning, composition, consistency, construction, and selection and applies them for architectural configuration items that is applicable in run-time. Generally, we present a suitable and understandable roadmap for applying changes in lower abstraction levels of architecture such as design and implementation levels.

In future, we will try to improve accuracy and reduce risks and probable errors by adding functionality of proper selection of architectural configuration items through fuzzy approaches.

REFERENCES


Razie Aldoosti received her MS degree in computer engineering from Sharif University of Technology, Tehran, Iran. She is currently a research assistant at Sharif University of Technology. Her research interests include software architecture, multi agent systems, machine learning, analyze method. She can be contacted at Performance Evaluation Software Engineering Lab, No. 601, Computer Engineering Department, Sharif University of Technology.

Shahrouz Moaven is a PhD candidate at Sharif University of Technology from 2009. He received his MS degree in software engineering at Sharif University of Technology, Tehran, Iran. His research interests include software architecture, software engineering, decision support system and business intelligence. He can be contacted at Performance Evaluation Software Engineering Lab, No. 601, Computer Engineering Department, Sharif University of Technology.

Jafar Habibi received PhD degree in computer science from University of Manchester in 1998. He is currently an associate professor and the chairman of Computer Engineering Department at the Sharif University of Technology, where he has been a faculty member since 1989. His research interests include software engineering, software architecture and design, performance evaluation, system analysis and design, information systems and simulation. He can be contacted at No. 626, the Department of Computer Engineering, Sharif University of Technology.

Mohammad Mehdi Khalili received his MS degree in computer engineering from Amirkabir University of Technology, Tehran, Iran. He is currently a research assistant at Amirkabir University of Technology. His research interests include software engineering, software architecture, computer networks, wireless sensor networks.