An Islamic application based on self-adaptive component

Meghnous Malak [1]
[1]Complex System Engineering Laboratory (LISCO), Department of Computer Science, Badji Mokhtar University, POB 12, 23000 Annaba, Algeria
MeghnousMalak@gmail.com

Atil Fadila
Complex System Engineering Laboratory (LISCO), Department of Computer Science, Badji Mokhtar University, POB 12, 23000 Annaba, Algeria
atil_fadila@yahoo.fr

Abstract

Self-adaptive systems are software systems that have to dynamically adapt their behavior in response to environment changes and users fluctuations. Thus, variability is a key concept in these systems; and handling it in early stages in the software development life cycle, helps to control complexity. The Design of these systems using component based architecture, enhances reuse and promotes dynamic reconfiguration, enabling system's modification without intercepting its execution, thus ensuring continuity of service. In this paper, we propose the dynamic adaptation of a component based Islamic application according to the user context (expert/novice), using orthogonal variability modeling and an aspect-oriented approach. We aim at combining UML Component Model and the orthogonal variability model in order to document variability. This allows to decrease system's complexity and to maintain whole view of the system's component based architecture and ensures variability traceability.

Keywords – *Self-adaptive system, Variability modeling, Orthogonal variability Model, Component based architecture, Aspect oriented paradigm.*

1. Introduction

Self-adaptive systems are software systems that have to dynamically alter their behavior in response to environment changes and user's fluctuations, e.g. system shifts from state N (configuration N) to a new state N+1 (new configuration N+1).

In: Proceedings of the 3rd Edition of the International Conference on Advanced Aspects of Software Engineering (ICAASE18), Constantine, Algeria, 1,2-December-2018, published at http://ceur-ws.org

Thus, modeling these systems is complex and crucial step in the development process, due to the large number of configurations and contexts.
Handling variability is a common activity between Software Product lines and self-adaptive systems. Hence, using SPLs mechanisms to model variability in self-adaptive systems will help to maintain the complexity.

Component based development is a paradigm in which the construction of applications is essentially based on assembly and reuse of existing entities called "software components". Thus, it promotes reuse and provides a component based architecture of system that can be configured and reconfigured by adding, altering or deleting components or connectors or interfaces.

The contribution of this paper is to construct an adaptive Islamic application according to the user context. The idea is to develop a component based architecture of the application to enhance component reuse and to benefit from the architecture reconfiguration. Also, and in in order to decrease system's complexity, we propose to model variability by using SPLs mechanisms, for instance, the OVM. Furthermore, to resolve variability at runtime and in order to guarantee separation of concerns, we propose an aspect oriented approach.

The remainder of this paper is structured as follows: section 2 and 3 introduce briefly self-adaptive systems and variability management respectively. Section 4 details our proposed approach, and illustrate it with a running example. Finally, section 5 concludes and addresses future works.

2. Self-adaptive systems

Dynamically adaptable software (DAS) self-adapts according to context information gathered from the surrounding environment [San17]. I.e. these systems alter their own behavior dynamically in response to

change in its operating environment, such as end-user inputs, external hardware devices and sensors, or program instrumentation [Sai09].

This behavioral change requires structural change, and the system shifts from state N to a new state N+1. Thus, a DAS needs to be able to sense its environment, to autonomously select an appropriate configuration and to efficiently migrate to this configuration. Handling these issues at the programing level proves to be challenging due to both the large number of contexts and the large number of software configurations which have to be considered. The use of modeling and the exploitation of models at runtime level provide solutions to cope with the complexity and the dynamic nature of DAS [Arc09].

Moreover, DAS can be considered as a software product line in which variability is resolved and bound at runtime. Hence, variability modeling techniques used in SPLs, could also be applied during design phase of DAS.

Designing such complex computing systems has been enhanced by the help of structural organization support of component-based architecture and model, in which software components encapsulates functionalities that can be accessed through well-defined interfaces that do not depend on their particular implementations. In addition to the benefits of modularity and reuse that result from this structural organization, adaptability and dynamic reconfigurability are key properties sought with this approach: it is possible to dynamically reconfigure the components assembly during runtime to cope with environment or user's fluctuations [Alv17].

Thereby, resolving variation points at runtime leads to dynamically reconfigure the component based architecture, by means of binding variants, e.g. adding, removing, replacing or altering components, connectors and interfaces.

In order to obtain a clear separation of concerns between the adaptation code and the computational code, we consider the management of variability as a non-functional requirement. Indeed, we use the aspect oriented approach to weave or unweave variants (in our work components) during runtime.

3. Variability management

Variability is usually understood as the ability of a software artifact to be changed (configured, customized, extended or adapted) in order to fit different contexts and environments. Variability can also be seen as the anticipated change [Gal14]. Variation has been largely addressed in the context of software product lines. Indeed, it is a key fact of most systems, if not all. In such systems, variation occurs for the same reasons as for the product lines [Hil10] [Gal11]; including:

I. Defer decisions about design or implementation.
II. Support multiple deployments.
III. Achieve system qualities such as adaptability.

Accordingly, handling variability (representing, managing, and reasoning about) in early stages of the software development lifecycle has become a relevant concern [Gal11].

3.1 FODA approach

Variability representation consists on capturing commonalities and differences between potential variants. In product line engineering, the FODA (Feature-Oriented Domain Analysis) model describes the visible properties of systems, in terms of product features. A feature is any stakeholder-relevant characteristic of a system [Hil10]. Indeed, the FODA model models such variability in a tree graph using variants (features that might be mandatory, optional or alternative) and variation points representing logical relationships between variants [Bos15].

On the basis of FODA, several extensions and enhancement of feature modeling and feature models were proposed, e.g. the FORM approach (Feature Oriented Reuse Method), that aim to enriching feature models with additional information, e.g., regarding feature cardinalities propositional constraints and non-functional properties of features [Hil10] [Bos15].

3.2 The orthogonal variability model

Modeling the variability within the traditional software development models has some significant drawbacks. For instance, it increases models complexity, because these techniques specify both common and variable parts of the system. Moreover, Variability informations cannot be traced if they are scattered among different models [Poh05] [Poh07].

The orthogonal variability model is a language and a methodology for superimposing variability over any software development artifact without interfering into its contents [Ecc16].

The OVM approach can be used to document the variability of arbitrary development artifacts, ranging from textual requirements to design models and even test cases. Also, the documentation of variability is separated from the technical realization of variability in the development artifacts and as such provides a further level of abstraction that aids developers in managing complexity and tracing variability [Poh07].

To that end, in our work we specified variability separately using the OVM as in figure 1, which depicts the graphical notations of this model.

4. The proposed approach

This section addresses the problem of developing an Islamic component-based application, that allows the inspection of food according to the Islamic percepts, and makes a verdict to classify them into Halal or not.

As early mentioned, the component based development provides, an entire view of the system which helps to reduce complexity; emphasize reuse; and promotes reconfiguration. Consequently, and due to its dynamic reconfiguration feature, we use the Fractal Component Model.

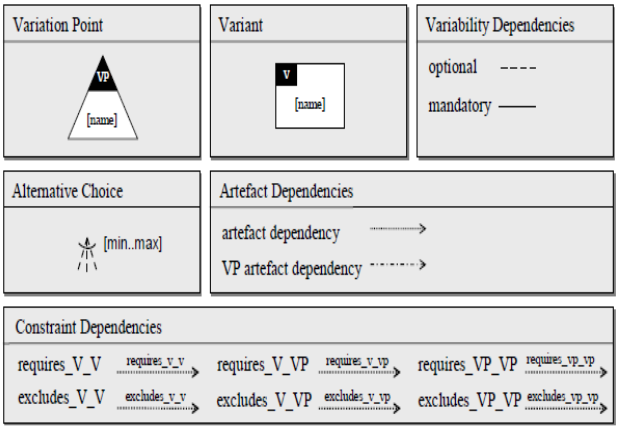

Figure 1: Graphical notation for variability models [Poh 05].

Moreover, our application is contextually adaptable, especially according to the user context, in order to offer different modes of operation, for users of different skill level (Novice user/ Expert User).

Variability management is a common key between software product lines and self-adaptive system. Thereby, several works have been established to identify synergy points and cross-fertilization opportunities between the two domains, such as [Alv12]. Correspondingly, we use the orthogonal variability model, which is a variability modeling mechanism in SPLs, to model information about variability in our system.

Furthermore, the dynamic adaptation of our architecture, which can be considered as the resolution of variation points, is implemented in AOP that proposes the separation of concerns, which allows the reuse of the same aspect in different components of the system. Thus, we use aspects to specify when, where, and how to reconfigure the system's architecture with variants

The following sub-sections detail each parts of our work.

4.1 Variability modeling and components design

We model the component based architecture with UML 2.0 Component diagram. In addition to, the Orthogonal Variability Model to document variability in design phase, in particular, to document our Islamic Component based architecture. This is depicted in figure 2, and table 1, which provides a brief description of composite components.

Also, we adopt the orthogonal variability modeling technique with a different semantic. Variation points indicate the containing relationship between components, e.g. TypeF Variation point indicates the potential subcomponents of the Food component (what subcomponents could be contained in the Food component), which are, the three variants: meat, additives, and drinks. The alternative choice with [1..3] cardinality, signify that the Food component must at least contain one subcomponent.

Table 1: Component description

Component name	**Description**
FoodClassification	Mandatory Component provides the name and the classification of the food (Halal/Haram).
Proof	Optional Component provides the proof about the classification of the food from the Holy Quran and the Sunna.

4.2 Adaptation of the architecture according to the context

Adaptation of the architecture, as shown in Figure 3, shifts the architecture of an application from a configuration (state N) to a new configuration (N + 1) according to a well-defined context, and following the adaptation policy.

4.2.1 Context representation

The context groups all the information about the running environment. In this work, we are interested in the user context, in particular, the expertise level of users.

As illustrated in figure 4, user context is characterized by, the current state of the architecture and, the incoming change of users which, in our work, is interpreted as events.

In our user context, we have two types of users:

i. A novice user: a user who does not need proof/evidence about the classification of the food.
ii. An expert user: a user who needs accurate and detailed proof and justifications about the classification of the food. This proof may be extracted from the Holy Quran or the Sunnah of the Prophet or both.

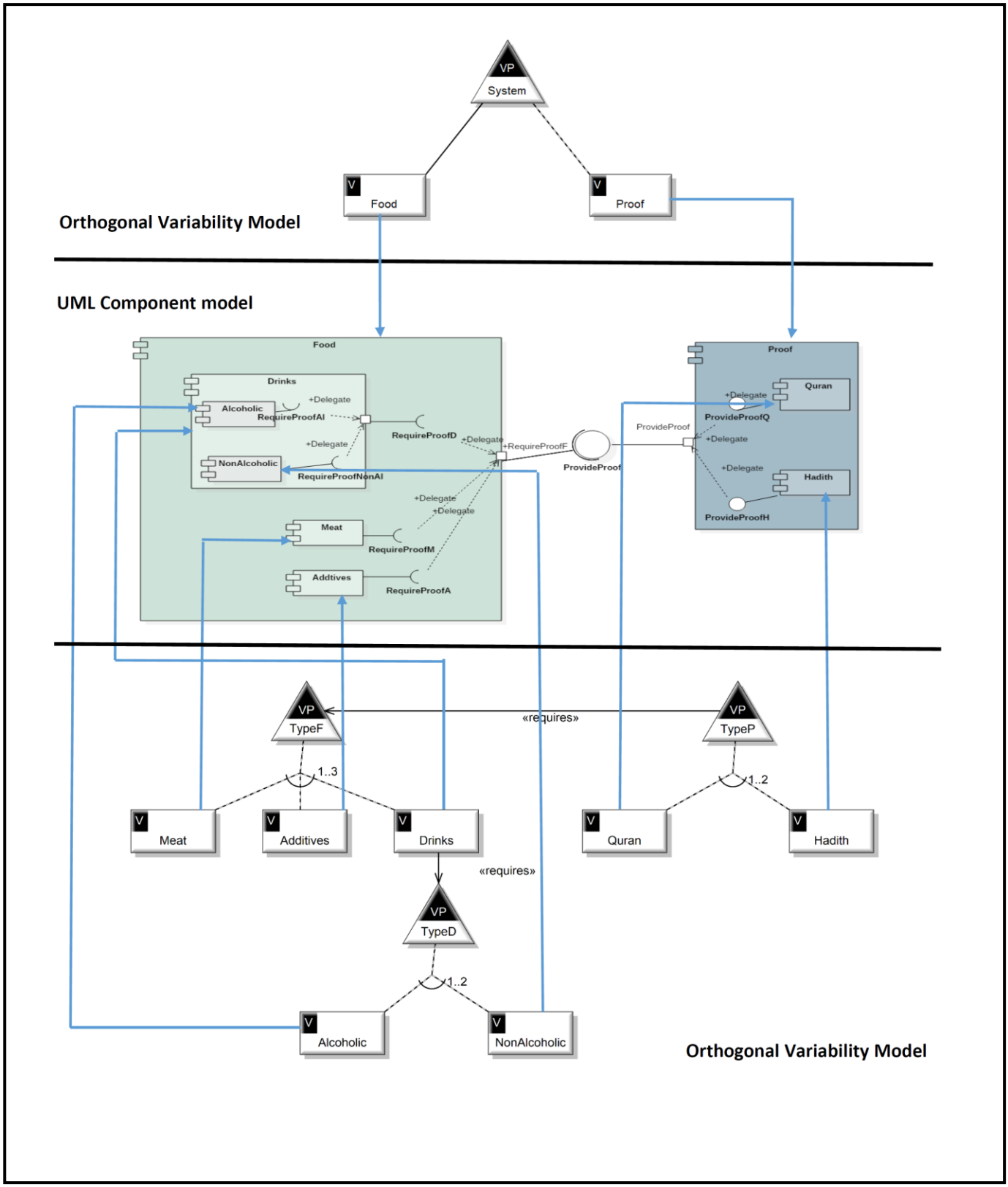

Figure 2: Linking UML Component Diagram with OVM for the Islamic Application.

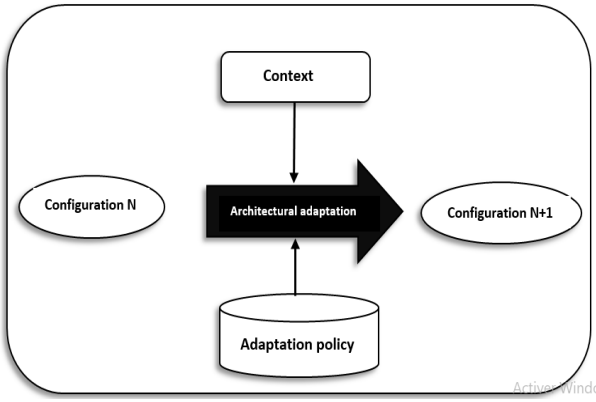

Figure 3: Description of the architectural adaptation.

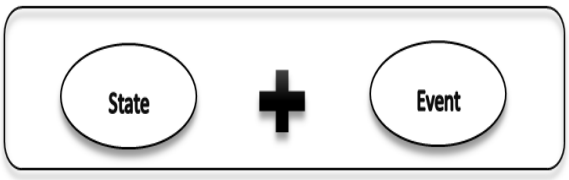

Figure 4: Context representation.

4.2.2 Adaptation policy

We have used a rule-based approach, specifically ECA (event, condition, action) rules, to resolve variability at both design and runtime.

The adaptation policy is a function that transforms a series of events that represent the context into a set of actions by respecting a set of conditions and starting from an initial state (configuration N).

In order to resolve variation point at the implementation level and provide dynamic adaptation, we have used the aspect oriented paradigm. It is a programming paradigm that allows isolating cross-cutting concerns, which are so-termed non-business functionalities (security, dynamicity). There are two main symptoms related to cross-cutting concerns: scattered and entangled code.

In our approach, the aspect intercepts and captures the state (configuration (N)) of the system as well as the event describing the context (Expert/Novice) (see figure 5). From these data and conditions (in the ECA rules), a series of actions (in the ECA rules) that serve to transform the system to a new state (an N + 1 configuration), are executed. Specifically, the aspect specify when, where, and weave/unweave components.

Lesson 1, illustrates an example of ECA-rule based adaptation policy.

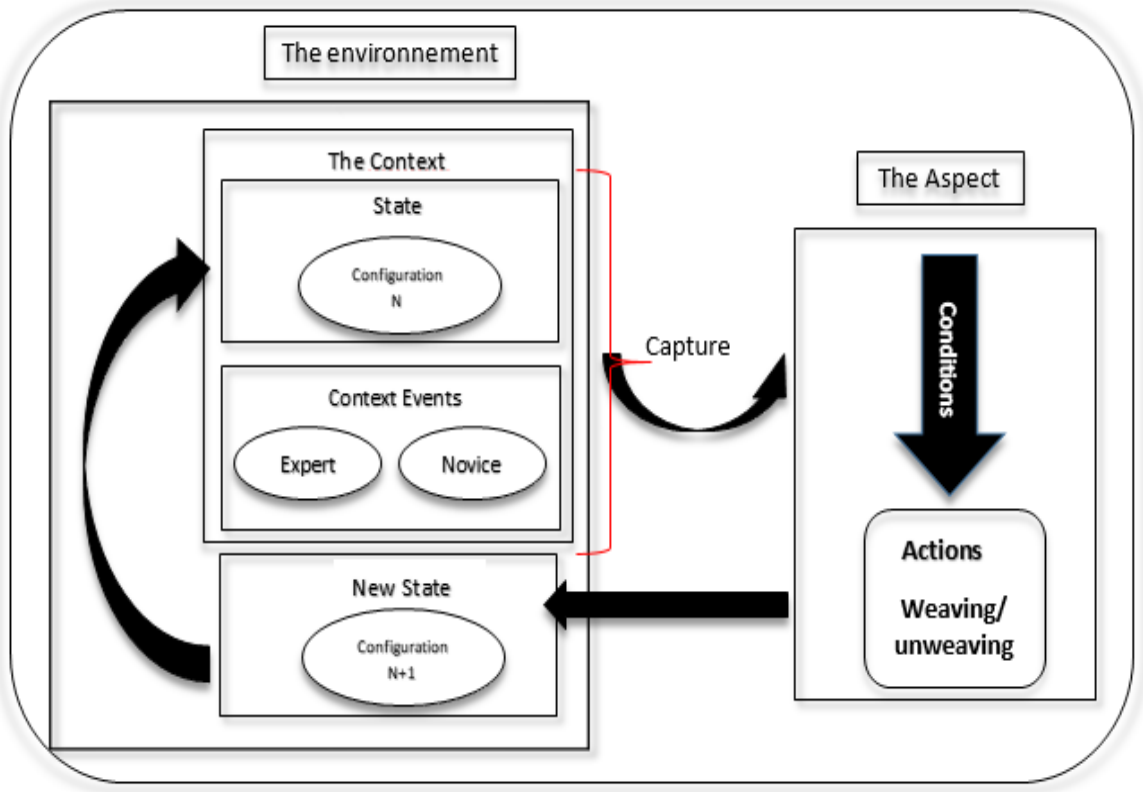

Figure 5: The adaptation policy with aspect oriented approach.

Lesson 1: Example of ECA-rule based adaptation policy.

Rule : Expert User:

Upon receipt of the "Expert User" event :

If the architecture does not contain the "Proof" component,

Then

If the user requests evidence from the Holy Quran

Then add the "Quran" component to the "Proof" component,

Else

If the user requests evidence from Hadith

Then Add the "Hadith" component to the "Proof" component,

Else

Add the "Quran" component and the "Hadith" component to the "Proof" component,

Add the "Proof" component to the architecture;

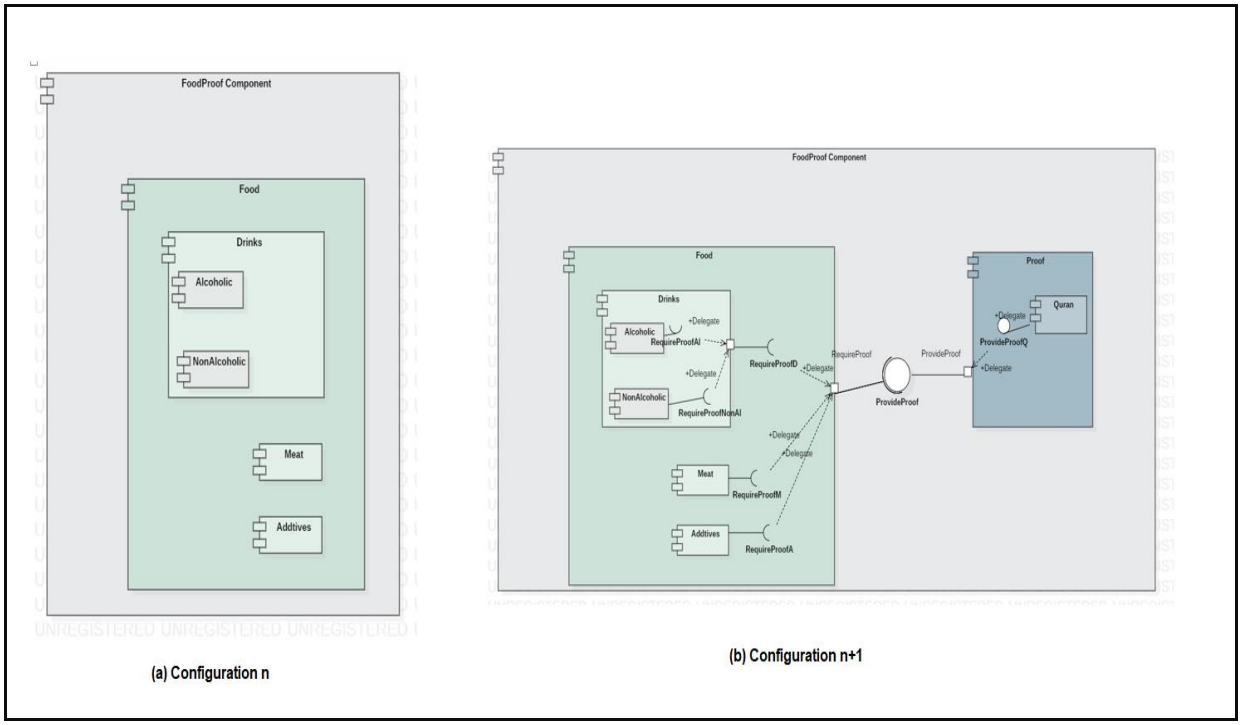

Figure 6: Dynamic reconfiguration of the system's architecture.

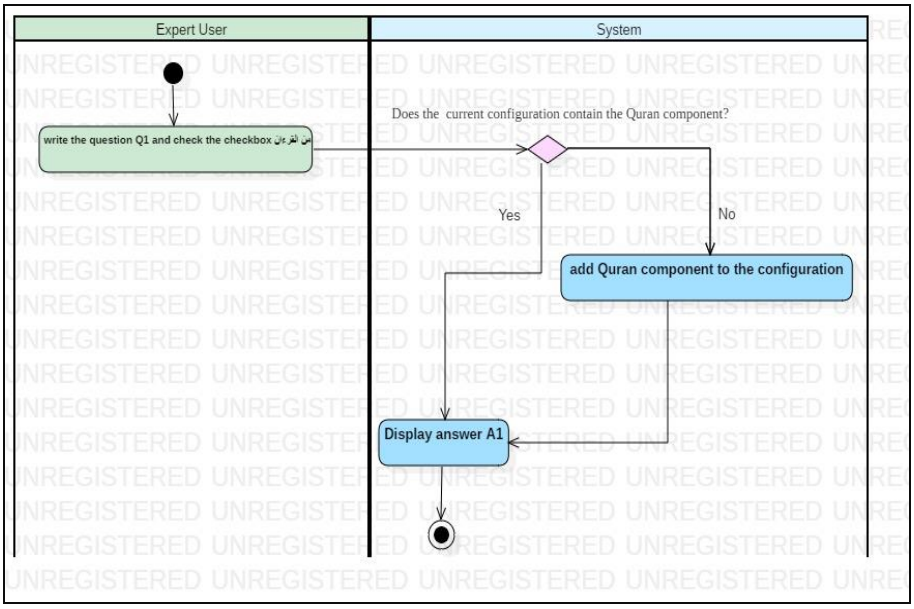

Figure 7: Activity diagram of the scenario.

In order to understand the adaptation, we propose the following scenario:

We suppose that the system's current configuration corresponds to figure 6.a.

1. The user writes the word "الخيرة" as the name of the food he wants to search its classification, and checks the checkbox " الخيل منالقرءان" in order to get evidence from the holy Quran about the classification of the searched word.

The system considers this user as expert (expert event), because he required evidence along with the answer.

2. The aspect checks if the current configuration of the system (figure 6.a) contains the Quran component:
   2.1 In the case, the answer is "yes": the system provides the answer.
   2.2 In the case, the answer is "no": the aspect adds the Quran component dynamically to the architecture (figure 6.b), which allows the system to provide the answer.
3. The system displays the answer which is : "نعم الخيرة حرام" along with the evidence

(number of surah=05 , number of verse=90).

In order to simplify the representation we consider: Q1 is the question asked by the user, and A1 is the answer displayed by the system.

Figure 7 depicts the corresponding activity diagram to the up-mentioned scenario.

5. Related work

Several researches have been contributed for runtime adaptation of systems. Similarly to our work, [Lou13] proposed an aspect oriented approach to dynamically adapt a component based system. However, variability is modeled implicitly by architectural aspect at the design level and no formal method to represent variation method has been used. Unlikely, in our approach, the variability is represented explicitly using the OVM.

In [Wol08], SPL engineering and plugin techniques have been combined to perform runtime adaptation. Based on the information documented in the variability models, and a plugin developed upon the .Net platform, runtime adaptation and dynamic reconfiguration are achieved. On the other hand,, our approach is based on components and aspect oriented paradigm that satisfy the separation of concerns property.

[Cos07], is another work that used the AOP combined with the computational reflection, to dynamically adapt the internal structure of aspect-oriented component. On the contrary, our approach achieve adaptation by reconfiguring the whole architecture. Moreover, the proposed adaptation has as purpose, correction or maintenance, while in our work, we focus on adaptation according to the user context. Finally, the variability modeling is not considered in this work.

6. Conclusion and future work

This paper addresses the adaptation of an Islamic application according to the user context. We have

modeled this application using a combination of UML Component Model and the orthogonal variability model in order to document variability. This combination reduces system's complexity and maintains a whole view of the system's component based architecture and ensures traceability of the variability.

Based on this view and the OVM model, we have proposed an aspect oriented approach to reconfigure the architecture in order to satisfy the user's needs.

Context representation in a formal way is one of the trends of self-adaptive systems. Our perspective is therefore to explore context variability representation.

7. References

[Alv12] V. Alves, D. Schneider, M. Becker, N. Bencomo, P. Grace, "Comparitive study of variability management in software product lines and runtime adaptable systems", Variability Modelling of Software-Intensive Systems, Sevilla, Spain, vol. 1, pp. 9-17, 2012.

[Alv17] F. Alvares, E. Rutten, L. Seinturier, "Domain-specific language for the control of self-adaptive component-based architecture", 1st ed., vol 130, Journal of Systems and Software, pp.94-112, 2016.

[Arc09] M. Acher et al., "Modeling context and dynamic adaptations with feature models", International Workshop Models at Run.time, Denver, USA, pp. 10, 2009.

[Bos15] J. Bosch, R. Capilla, R. Hilliard, "Trends in systems and software variability", 3rd ed., vol. 32, IEEE Software, pp.44-51, 2015.

[Cos07] C.Costa, J.Pérez, J.Á.Carsí, "Dynamic adaptation of aspect-oriented components", Component-Based Software Engineering, Medford, USA, vol.1, pp.49-65, 2007.

[Ecc16] J. Echeverria, F. Pérez, C. Cetina, O. Pastor, "comprehensibility of variability in model fragments for product configuration", Conference on Advanced Information Systems Engineering, Ljubljana, Slovenia, vol. 1, pp. 476-490, 2016.

[Gal11] M. Galster, P. Avgeriou, "Variability in software architecture: current practice and challenges", 5th ed., vol. 36, ACM SIGSOFT Software Engineering Notes, pp.30-32, 2011.

[Gal14] M. Galster, D. Weyns, D. Tofan, B. Michalik, P. Avgeriou, "Variability in software systems— a systematic literature review", 3rd ed., vol.40, IEEE Transactions on Software Engineering, pp.282-306, 2014.

[Hil10] R. Hilliard, "On representing variation", ECSA 2010 workshops: Workshop on Variability in Software Product Line Architectures, Copenhagen, Denmark, vol. companion, pp.312-315, 2010.

[Lou13] S. Loukil, S. Kallel, M. Jmaiel, "Runtime adaptation of component based systems", Networked Systems, Marrakech, Morocco, vol. 1, pp. 284-288, 2013.

[Poh05] K. Pohl, G. Böckle, F.V. Linden, "Software product line engineering - foundations, principles, and techniques", Springer, 2005.

[Poh07] K. Pohl, A. Metzger, "Variability management in software product line engineering", International Conference on Software Engineering, Minneapolis, USA, vol. companion, pp. 186-187, 2007.

[Sai09] M. Salehie, L. Tahvildari, "Self-Adaptive software: landscape and research challenges", 2n ed., vol. 4, ACM Transactions on Autonomous and Adaptive Systems, pp.1-42, 2009.

[San17] I. Santos, T.A. Oliveira, E.S. Almeida, R.M. Andrade, "Dynamically adaptable software is all about modeling contextual variability and avoiding failures", 6th ed., vol. 34, IEEE Software, pp. 72-77, 2017.

[Wol08] R. Wolfinger, S. Reiter, D. Dhungana, P. Grunbacher, H. Prahofer, "Supporting runtime system adaptation through product line engineering and plug-in techniques", International Conference on Composition-Based Software Systems, Madrid, Spain, vol. 1, pp.21-30, 2008.