

REQUIREMENTS INTERACTIONS MANAGEMENT: A MULTI-LEVEL FRAMEWORK

Mohamed Shehata Armin Eberlein
Department of Electrical & Computer Engineering
University of Calgary
2500 University Drive NW
Calgary, AB, T2N 1N4 Canada
{Shehata; Eberlein}@enel.ucalgary.ca

ABSTRACT

The development of software based on reusable requirements has received more attention lately because of the benefits it can potentially provide for software development [1], [2]. However, this type of development is not an easy task to do. It is our belief that such development must be done in conjunction with careful management of the relations and dependencies that can arise from combining requirements of already existing components. This paper deals with requirements interaction management at different levels of complexity. We present a framework for the detection of requirements interactions at three levels. The first level of this framework can be used in situations where information about known interactions is already available in a knowledge base. The second level identifies likely (but not guaranteed) interactions between requirements based on requirements attributes. The third level requires the development of formal models of the system, which is very time consuming but very efficient in the detection of feature interactions. The main purpose of this framework is to help developers detect interactions between requirements at different levels of complexity in a cost-effective manner.

KEYWORDS

Software requirements, reusability, requirements interaction management, requirements attributes.

1. INTRODUCTION

Reuse is a well-known practice in software development. The attraction of software reuse comes from the possible improvement it can provide to software development productivity and quality. It also has the potential of reducing the development costs and the product's time to market [1], [2]. Reusability can be introduced at different phases in the software life cycle. Most research so far has only addressed reuse at the design and implementation levels. At both levels,

remarkable results have been achieved, e.g. with algorithms at the design level and with code libraries at the implementation level. However it is argued that reusability at the requirements level can drive even more reusability at the design and implementation levels [3]. Therefore, requirements reusability should receive more attention to exploit all its potential benefits.

Through our review of the current research on reuse at the requirements level, we were able to identify a common procedure that was in some form or another adopted in various approaches that we had reviewed. This procedure helps us understand how to create a reusable platform and how to create a new system from that platform.

The first part of this procedure starts with the analysis of a group of systems in a certain domain that have numerous commonalities between them, i.e., it is promising to build a reusable platform for them (e.g. systems that are part of a product line).

The second part of this procedure is the actual building of a new system that belongs to this specific domain. The developer will reuse requirements of the common platform to build the new system. Also, the developer has to introduce some new requirements, which this specific system needs. The aim is that the reuse of requirements together with their implementations helps develop a complete specification in less time and with less cost.

It is our belief that this type of development must be done in conjunction with the careful management of the relationships and dependencies that arise between reused and new requirements. Requirements Interaction Management (RIM) was discussed in detail by W. Robinson in 1999 and he defines it as "the set of activities directed towards the discovery, management, and disposition of critical relationships among a set of requirements" [3]. It is very similar to feature interaction management used in the telecommunication domain in that they both try to detect possible interactions between features or components and provide

guidance on how to resolve these interactions. However, so far research in feature interaction is mainly concerned about telephony features and has a very narrow view of requirements. In addition, feature interaction in a telephony system has a well-defined domain and the requirements of this domain are well understood and researched.

In order to understand the increased importance of requirements interaction management in the context of requirements reuse, we have to consider that there are parameterized requirements whose parameters need to be assigned specific values, which can cause interactions. Additionally, there are new requirements that will be added to the system and might negatively interact with the already existing set of requirements. Therefore, the question is how to ensure that the values that were assigned to the parameterized requirements or the new requirements that were added will not interact with the already existing requirements? Of course there is no easy answer and it is likely that one has to continually manage the requirements for any interactions that might occur.

The traditional way of detecting requirements interactions is done by domain experts or by using formal modeling. However, the first method is error prone while the latter method is very expensive and time consuming.

In this paper, we introduce a general framework to detect requirements interactions at three different levels. The developer can use any one of them or even a combination of them based on

his/her needs. Therefore, we believe that this framework is very useful as it provides guidance on how to detect interactions between requirements ranging from inexpensive and quick methods, as in the first level of the proposed framework, to more complex approaches to interaction detection, as in the third level of the proposed framework.

The rest of this paper is structured as follows: Section 2 gives details on our proposed framework for the detection and management of requirements interactions. Section 3 summarizes our conclusions and outlines our future plans for the enhancement and the application of this framework in an industrial case study.

2. PROPOSED FRAMEWORK TO REQUIREMENTS INTERACTION MANAGEMENT

Our aim is to develop a framework that supports the detection of requirements interactions at different levels of complexity.

The first level of this framework can be used for the detection of already known interactions which are stored in a knowledge base. The second level can be used to locate likely interactions between requirements based on attributes of the requirements. The third level requires the development of formal models of the domain, which is very time consuming but very good for the detection of feature interactions.

A major benefit of defining these levels is the ability to detect simple interactions in the first

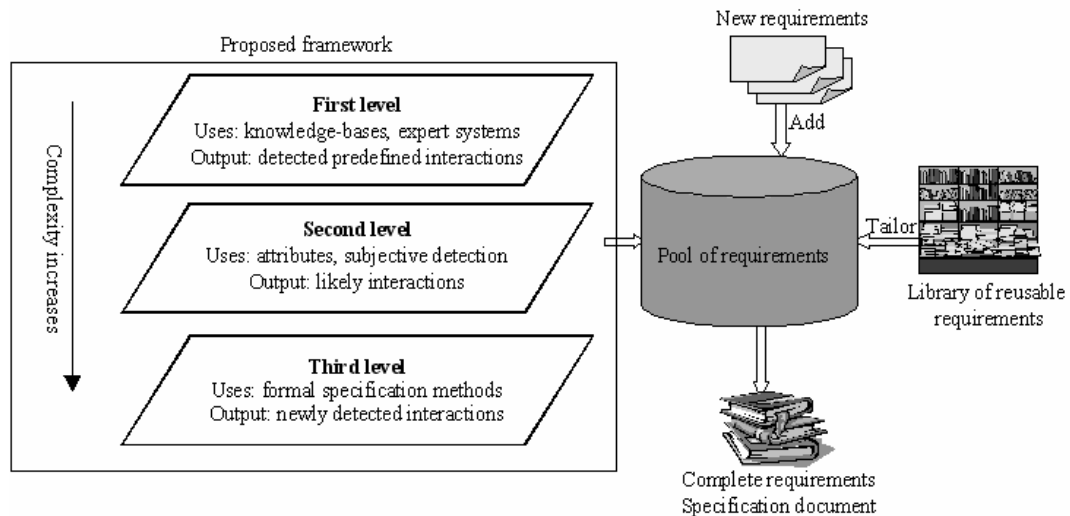


Figure 1: The proposed framework overview

level at very low cost. Depending on the type of system, it might be necessary to go to the second or even the third level. This way the developer will be able to detect most of the interactions very early without the need to use formal methods which are very expensive and time consuming. However, our framework still supports an in-depth analysis using formal methods, for instance to be used for mission-critical systems. Figure 1 shows a general overview of our proposed framework. The procedure starts when a new system is created (or a requirement is added/changed in an already existing requirements specification). The developer uses a library of reusable requirements to select a set of requirements and tailors them to the new system-specific needs (e.g. [4]). Then the developer adds some new requirements that the new system requires to form a complete set of requirements. At this point in time, the requirements are not yet checked for interactions. Then our framework is used to detect and resolve unwanted interactions. In the next subsections we introduce the three levels of our proposed framework in more detail.

2.1 THE FIRST LEVEL

The first level of this framework can be used in detecting interactions that are already known and stored in a knowledge base. This will be done using artificial intelligence techniques like expert systems. The basic idea here is to capture from domain experts all the interactions that are known and then store these interactions in the knowledge base of an expert system. The limitation of this level is that it will only detect those interactions that are already known to occur in certain scenarios. For instance, the two requirements “high reliability” and “low cost” negatively interact with each other and there must be some sort of resolution or trade-off to handle this interaction. This example illustrates the detection of known requirements interactions. Many of the interactions that can be detected by this level are interactions between quality requirements. Quality requirements are very well researched and the interactions between them are well defined. Moreover resolutions for these interactions have been suggested in the literature [5].

There is also another type of known interactions that can be detected by this level, which are interactions that are known to happen between certain requirements in a certain domain. For example, let us consider a certain domain like the telecommunication domain, which has numerous examples of known feature interactions. One

common feature is Call Forwarding on Busy Line (CFBL), which is defined as “a telephone-company-activated feature that forwards incoming calls to a subscriber to another line when the subscriber line is busy”. Another common feature is Call Waiting (CW), which is defined as “a feature that informs a busy subscriber that another call is waiting by playing a tone” [6]. Obviously, these two features will negatively interact as they cannot be activated at the same time.

Three important issues must be considered when using this level of detection. The first issue is the collection of known feature interactions. The second issue is the construction of the knowledge base that contains the known feature interactions. The knowledge base must be very well structured and contain all the interactions that are known. The third issue is the necessity to continuously update this knowledge base with new interactions that will be discovered in the future.

2.2 THE SECOND LEVEL

The second level of our framework can be used to detect likely interactions between requirements based on their attributes. This second level indicates only likely interactions and cannot guarantee the detection of all interactions. The basic idea used in this level is to define domain attributes and to look out for those combinations of these attribute values that are likely to result in requirements interactions.

Table 1 shows some attributes that we will use to illustrate the usage of this level to detect interactions.

First we start by explaining that any one of the table attributes (informal definition, current state, activation, actor, etc) is optional and can either contain a value or an N/A to indicate that this attribute is not applicable to this requirement. Also, depending on the domain, the developer can add different attributes as columns in the table. Very different kinds of requirements can be represented in this table. However, we anticipate further modifications to our current list of attributes as we use more examples.

In the following we describe scenarios that result in likely interactions between two requirements.

(I) First scenario:

The current state attributes of both requirements are N/A and their activation attributes are the same.

Explanation: Some requirements define dynamic behavior that the system should exhibit when a certain action occurs. This scenario detects interactions when there is a certain event that

activates two requirements simultaneously. Unfortunately, there is a possibility that these two requirements interact if activated at the same time. For example, in the first two requirements in Table 1, the event “pressure goes below atmospheric pressure” occurs and requires a certain dynamic behavior to be executed by the system as described by the first two requirements. The first requirement requires the system to increase the cockpit pressure, while the second requirement requires the system to post a warning message. Therefore, the dynamic behavior required to fulfill the first requirement might contradict the dynamic behavior required to fulfill the second requirement. But after a closer look at the actions, we see that there is no negative interaction between increasing the pressure and posting a warning message and therefore the two dynamic behaviors required will not contradict each other. This example is introduced here to show that this level will only indicate likely interactions but not guaranteed interactions (because it indicated that there is a possible interaction between the two requirements but after analysis of the required actions the decision was made that the required actions will not contradict each other and therefore there is no interaction).

(II) Second scenario:

The activation attributes of both requirements are the same and the current state attributes of the requirements are also the same.

Explanation: It is the same as the first scenario except that the current state of the system is known, i.e., this scenario is a special case of the first scenario. In this case now, the current state of the system must be taken into account and should be the same. Requirements three and four in table 1 are examples for this case: The current state of the system is that the line is busy and then there is an incoming call request. One requirement then defines that the incoming call be forwarded to another line. However, the other requirement states that a tone be played to inform the user of the incoming call request. Therefore, the two requirements contradict each other leading to a negative interaction.

(III) Third scenario:

The feature-involved attributes for both requirements are the same, the activation attributes are N/A, and the current-state attributes are N/A.

Explanation: If two requirements have a common feature that describes the behavior of the system to fulfill these two requirements, then there is a likely interaction between the two

requirements. An example can be seen in table 1 (requirements 5 and 6). Both requirements have a common feature that describes the implementation of these two requirements as part of the system office tools.

(IV) Fourth scenario:

The feature-involved attributes of both requirements are the same and the activation attributes are the same.

Explanation: This is a more specific case of the third scenario except that the activation attributes of the two requirements are known. In this case, these activation attributes must be the same to achieve the condition of simultaneous occurrence of the two requirements. This is the case with requirements four and seven in table 1

(V) Fifth scenario:

The resource consumption of the first requirement and the resource consumption of the second requirement cause a conflict. Conflict here can mean exceeding a predefined limit on a certain piece of hardware or it can mean that they both require the usage of the same resource at the same time in a conflicting manner.

Explanation: If two requirements have the same activation and there is a chance that both are activated at the same time, they might violate resource constraints, which can cause negative interactions. Requirements four and eight in table 1 give an example: When the user tries to activate call waiting and 3-way calling at the same time, both of them use a certain piece of hardware called bridge. However, there is only one bridge for each agent and this bridge can support only one feature. Therefore, they both require the usage of the same hardware at the same time which is not feasible. And as a result, the two requirements will negatively interact with each other [7].

The developer starts by formulating all requirements in this table and assigns appropriate values for the different attributes. If one attribute does not apply to a certain requirement, it gets the value N/A. Also, during the development of this table, a data dictionary should be used to record any terminology that has more than one meaning or different terminologies that have the same meaning. Finally, the evaluation of interactions is done subjectively by experts that can identify interactions based on the five scenarios defined above.

2.3 THE THIRD LEVEL

The third level is used to detect interactions between requirements that have not yet been identified by any of the other two levels. Although this level is expensive and time

consuming, it is very good in the detection of feature interactions. The basic idea here is to develop formal models of the domain and of the new system using a formal specification language, and then to validate these models. There are various ways how this can be approached. Telecommunication companies have been successful in the detection of feature interactions using formal specification languages, such as SDL [8]. We are not yet working on this level in our research, however, our initial thoughts are directed towards the representation of each requirement in an event-action table, where the attribute activation is considered as an event and the attribute action is the action, as explained in section 2.2. Then we can use temporal logic and first order logic to represent each requirement in a mathematical form. If two requirements have the same activation but different actions, then these two requirements will interact with each other. For example, let us consider a lift system that has the following two requirements [9]:

Req₁: “When the lift door has been opened, it will close automatically after d time units”.

Req₂: “When the lift has stopped, it will open the door.”

Let’s suppose that we add the following requirement:

Req₃: “When the lift stops, the door will stay opened until a call is made from floor k”.

The event of Req₁ is “door opened” and the action of Req₁ is “close automatically after d time units”. Also, event of Req₂ is “lift stops” and action of Req₂ is “open the door”. Finally, event of Req₃ is “lift stops” and the action of Req₃ is “door will stay opened until a call is made from floor k”. Using logic, we can conclude from requirements 2 and 3 that when the door is open it will stay opened until a call is made from floor k. This new conclusion can be stated as Req₄ which has the event “open door” and the action is “doors will be opened until a call is made from floor k”. Obviously this will negatively interact with requirement 1. Generally we can say that an interaction is detected whenever:

Req₁(event) \cap Req₂(event) $\neq \emptyset$ and Req₁(action) contradict with Req₂(action).

3. CONCLUSION

In this paper we introduced a framework to detect interactions at three different levels of complexity. The first level can be used to detect

predefined interactions that are stored in a knowledge base. The second level can be used to detect interactions that are likely to happen. In the third level we suggest using a formal domain description to achieve full interaction detection. Although this framework is not yet fully developed, we believe that it represents a good step towards the formulation of complete and interaction-free systems. Our future work will be to complete the third level in this framework and to validate it in an industrial case study.

REFERENCES

- [1] Lim, W.C. (1994) Effects of Reuse on Quality, Productivity, and Economics, IEEE Software, 11(5): 23-30,1994.
- [2] Sommerville, I. (1996) Software Engineering (5th Edition), Addison Wesley, ISBN 0-201-42765-6.
- [3] Robinson, W.N., S. Pawlowski, Volkov, S., Requirements Interaction Management, Georgia State University, Atlanta, GA, August 30, 1999.
- [4] Mike Mannion, Hermann Kaindl, Joe Wheadon, Barry Keepence, (1999), Reusing single system requirements from application family requirements. Proceedings of the 21st international conference on Software engineering May 1999
- [5] Barry Boehm, Hoh In, Identifying quality-requirement conflicts, IEEE Computer Society, March 1996 (Vol. 13, No. 2)
- [6] A. Felty and K. Namjoshi. Feature specification and automatic conflict detection. In Calder and Magill [1], pages 179-192.
- [7] Kenneth H. Braithwaite and Joanne M. Atlee. Towards automated detection of feature interactions. In “Feature Interactions in Telecommunications systems” IOS Press, Amsterdam, 1994. pages 36-59.
- [8] ITU-T Recommendation Z.100 Annex F: SDL Formal Semantics Definition, International Telecommunications Union (ITU), Geneva, 2000
- [9] Marrita Heisel and Jeanine Souquières, Detecting feature interaction - a heuristic approach. In G. Saake and Can Türker, editors, Proc. Of the first FIREworks workshop, Preprint 10/98, pp. 30-48, Fakultät für Informatik, 1998, Univ.Magdeburg.

Informal Definition	Current state	Activation	Actor	Action	Feature involved	Resource consumption
The system shall increase the cockpit pressure if the pressure inside the cockpit goes below the atmospheric pressure	N/A	Pressure goes below atmospheric pressure	System	Increase cockpit pressure	N/A	N/A
The system shall post a warning message if the cockpit pressure goes below the atmospheric pressure	N/A	Pressure goes below atmospheric pressure	System	Post warning message	N/A	N/A
The telephone company shall forward incoming calls to a subscriber to another line when the subscriber line is busy	Line is busy (Subscriber X is connected to subscriber Y)	Incoming call request	Telephone company	Forward incoming calls to another line	CFBL is active	N/A
The system shall use CW to inform a busy subscriber that another call request is waiting by playing a tone...etc	Line is busy (Subscriber X is connected to subscriber Y)	Incoming call request	System	Play a tone to inform user	CW is active	Bridge (A piece of hardware)
The system shall use word XP as word processor as part of the office tools	N/A	N/A	System	Use word XP	Office tools	N/A
The system shall use EXCEL 2000 as a spread sheet as part of the office tools	N/A	N/A	System	Use EXCEL 2000	Office tools	N/A
The system shall use CW to inform an off-hook telephone that there is another call request by playing a double click tone ...etc	Line is Off-hook	Incoming call request	System	Play double-click tone to inform user	CW is active	N/A
If the user subscribes to 3-way calling and this feature is enabled when the line is busy, then If he receives a call request, this feature will allow him to talk to both callers at the same time.	Line is busy (Subscriber X is connected to subscriber Y) and 3-way calling feature is active	Incoming call request	N/A	Allow the user to talk to both callers at the same time	3-way calling	Bridge (A piece of hardware)

Table (1): different requirements attributes