APPLYING AN EVENT-BASED APPROACH FOR DETECTING REQUIREMENTS INTERACTION

Edgar Sarmiento, Marcos R. S. Borges and Maria Luiza M. Campos
Graduate Program in Informatics, Federal University of Rio de Janeiro, Brazil

Keywords: Requirements Engineering, Requirements Interaction Detection, Interaction Type.

Abstract: At the software development cycle, it is in the requirements analysis phase that most of the problems that can compromise the delivery time and the development and maintenance costs must be identified and resolved. In general, the requirements obtained in this phase have different relationships with each other. Some of these relationships, commonly called negative interactions, make difficult or impossible the progress of some activities of the development process. The detection of interactions between requirements is an important activity that may prevent some of these problems and avoid their propagation throughout the remainder activities. Most of the existent research in this area only focuses on the requirements phase, mainly in the identification of conflict and/or inconsistency interactions. This paper presents a semi-formal event-based approach to model and identify the interactions between requirements, investigating the interactions that influence the other phases of the software development process.

1 INTRODUCTION

Research in requirements engineering has demonstrated that the requirements of a system are not usually independent from each other. In fact, there are different types of interactions between them (Dahlstedt and Persson, 2003; Robinson et al., 2003; Shehata, 2005). This happens because the different elements that compose a system are not isolated entities: the relationships and interactions among these entities make possible the functioning of the system.

Commonly the set of requirements of a complex system have interactions. Among these interactions exists a set of negative interactions (i.e. conflicts or inconsistencies), which must be identified and resolved in the initial phases of the development cycle. The interactions identification is an important activity that allow: to resolve eventual conflicts, to better plan the requirements implementation, to manage the impact of a change on other requirements, and to plan the tests considering the interactions. Several researches show these benefits, however, most of them only focus on the requirements phase (Dahlstedt and Persson, 2003; Robinson et al., 2003).

The Requirement Interaction Problem was introduced and well researched in the telecommunications domain, with the name of Feature Interaction Problem (Calder and Magill, 2000). The problem of features interaction is generally understood as a situation where the integration of several features in a system could interfere or affect one another. However some problems were identified on these methodologies. Most of them focused on the telecommunications domain, their use is not known in the software domain and most of them use formal specification languages, if used correctly they are accurate, although difficult and expensive of using. Requirements Interaction Management was introduced by Robinson et al. (2003), defined as the set of activities directed towards the discovery, management, and disposition of critical relationships among a set of requirements. Requirements Interaction is similar to Features Interaction, however, Features Interaction only considers functional requirements.

Several methods for the identification of interactions on the requirements analysis phase have been proposed. Lamsweerde et al. (1998) and Robinson et al. (2003) gave an extensive review on the different interaction types. They focused on how to identify and eliminate negative interactions. As most of these approaches, they are based on formal specification languages. Mylopoulos et al. (1999)

presented an approach to discover interactions between functional and non-functional requirements based on dependencies graphs built in a hierarchical way. And among the researches that focus on the design phase, we can mention those presented by Yuqin et al. (2006) and Zhang et al. (2006). These approaches are based on features. A feature is defined as a set of cohesive requirements. These approaches consider mainly dependence interactions (positive interactions). For their correct functioning it is necessary to identify conflicting or inconsistent interactions in previous phases.

Among those works more related to ours we can mention IRIS (Shehata, 2005) and BPA (El-Ansary, 2002). IRIS is a semi-formal approach to detect requirements interactions. This approach uses tables and graphs together with scenarios of interaction. A system is considered a compound of static and dynamic requirements and their resources, each one of these elements consisting of attributes.

BPA (Behavioural Pattern Analysis) is an approach in which events are considered as primary entities of the models of the world. This approach proposes the representation of requirements based on the actions and events of the system, and the relationships between them.

This paper proposes an approach for specifying and identifying the different interactions between requirements, using a semi-formal method based on events. It is based on events, because the flow of events describes the behavior of the system through a set of interactions between objects. Our approach considers the most interaction types described in the literature and supports the identification of interactions with less effort and complexity.

This paper is structured as follows. Section 2 presents the interaction definitions, the attributes used for specification and subsequent requirements interaction identification. Section 3 presents the interaction types considered in our approach and the detection rules for each type. Section 4 presents the proposed approach. Section 5 presents an example of application of the approach. Finally, in Section 6 we present our conclusions so far and the directions for future work.

2 REQUIREMENT INTERACTION

There is an interaction when two or more requirements have some effect on each other. These interactions can be caused by different viewpoints of stakeholders, change or re-use of requirements, component-based development, among others. Some definitions of interactions can be found in (Dahlstedt and Persson, 2003; Robinson et al., 2003).

2.1 Basic Attributes of Requirements

The dynamic models in the Object Oriented Approach represent the behaviour of the system, i.e., the interactions among the different objects of the system and their environment. These interactions are caused mostly by the presence of events produced by another object or some external entity. These events stimulate and control the functioning of the system.

The identification of events during the requirements analysis phase allows to reveal the different interactions that exist among the set of requirements, and subsequently, to know the set of interactions among the different objects. The events are important because the implementation of a requirement produces a result (event) and the events cause the implementation of a requirement.

In our approach a requirement is described and represented as a set of attributes that consists of: events, action, states and resources. (Table 1)

Some of the concepts presented in this section were extracted from (BPMN, 2006) and (El-Ansary, 2002; Shehata, 2005).

Table 1: Attributes of Requirements.

Attribute	Description
IDR	The identifier of the requirement.
Description	The description of the requirement
Event	The incidents or facts that happened inside or outside an object.
Action	The activities carried out during the execution of the requirement, such as calculations, generation of events, etc.
Object	The objects involved in the execution of the requirement.
Resource	The instruments or tools used by the requirement to complete its execution.

2.2 The Role of Events in Requirements Interaction

To investigate the possible roles of the events in the set of interactions among requirements, it is necessary to understand the definition and the description of each event. Based on this observation, we identify the need for specifying an event using attributes. For this, each requirement has a set of associated events, and it is necessary to specify each one of them (Table 2).

Table 2: Attributes of Events.

Attribute	Description
IDE	The identifier of the event.
Description	The description of the event.
IDR	The identifier of the requirement.
Type	Message, Time, Rule, Link, Multiple and Cancel (BPMN, 2006).
Category	*Input*: it stimulates some action. *Output*: it's generated by some action.
Action	The Action that produces or is stimulated by the event.
Object	The event causes changes of state of objects: Pre-state and Next-state.
Resource	The resources stimulated by the event.

Figure 1 illustrates the role of the events and actions in the interactions among requirements. Requirement R1 executes action R1A1; R1A1 produces (output) the event R1E1. In another side Requirement R2 executes action R2A1; R2A1 is stimulated by R1E1 (input).

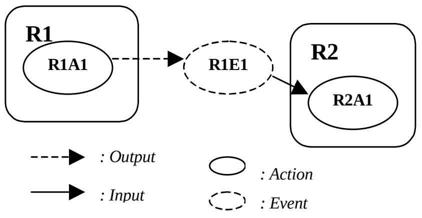

Figure 1: The Role of the Events.

3 INTERACTION TAXONOMY

Among these interactions it is possible to identify two types of general interactions, positive and negative ones. Positive interaction types are relationships of intrinsic dependence (Requirement R1 requires R2), while negative interaction types mainly include conflicting interactions.

Numerous classifications were generated to represent requirements interactions types (Dahlstedt and Persson, 2003; Shehata, 2005; Yuqin et al., 2006; Zhang et al., 2006; Pohl, 1996). Lamsweerde et al. (1998) present an extensive review aboutinconsistency interactions. The classification presented in this work was generated considering the types founded in the literature. We considered the basic types that have a significant effect in the remainder of the software development process, especially in the software design phase. A more detailed description of the different interaction types is showed in Table 3 and Table 4.

Consider the requirements ***Rx*** and ***Ry***. ***Rx*** is different of ***Ry*** and ***RC*** is a resource.

Table 3: Negative Interactions.

Negative Interaction	Description
Conflict	Set ***Rx*** and ***Ry***, there is a condition ***B*** (event) that can cause a conflict.
Cancel	When the execution of ***Rx*** overrides and cancels the execution of ***Ry***.
Negative Impact	When the execution of ***Rx*** overrides, but it does not cancel the execution of ***Ry***.
Resource Conflict	***Rx*** and ***Ry*** simultaneously access to the same resource ***RC***.
Resource Blocking	When the execution of ***Rx*** takes completely and blocks ***RC***.

Table 4: Positive Interactions.

Positive Interaction	Description
Require	***Rx*** requires ***Ry*** for functioning.
Inform	***Rx*** sends a piece of information to ***Ry*** to indicate that certain condition was reached.
Configure	***Rx*** configures ***RC*** that is accessed by ***Ry***.
Flow	***Rx*** processes data that is passed to ***Ry*** through a flow (Flow).
Collateral	***Rx*** and ***Ry*** are simultaneously activated to complete the execution of ***Rz***.

3.1 Interaction Detection Rules

For each one of the interaction types shown in Table 3 and 4, a set of rules for interaction detection was defined and created. These rules involve each of the attributes defined in Tables 1 and 2.

All the rules identified were built based on the template:

WHEN <Event>

IF <Pre-condition>; it matches the objects, events, actions and resources of two requirements.

THEN <Interaction Type>

Table 5: Require Interaction.

Interaction Type	Require
Description	When the **Event** produced in **Rx** stimulates the **Action** executed in **Ry**. Requirements: **Rx** and **Ry**; **Rx** != **Ry**.

Rule	**WHEN** Event
	IF {(Rx.Event_X = Event **AND Ry**.Event_Y = Event)
	AND (Ry.Event_Y.Category = Output)
	AND (Rx.Event_X.Category = Input)
	AND (Ry.Event_Y.type != Cancel) **}**
	THEN Rx Require Ry

In Tables 5 and 6, we present some of the rules identified for some of the interaction types. The other rules are listed in (Sarmiento, 2008).

Table 6: Cancel Interaction.

Interaction Type	Cancel
Description	When the **Event** that stimulates **Rx** cancels the **Action** executed in **Ry**. **Rx** and **Ry** have the same **Action** and **Object**. Requirements: **Rx** and **Ry**; **Rx** != **Ry**.
Rule	**WHEN** Event **IF{(Rx**.Event_X = Event **AND Ry**.Event_Y **!=** Event) **AND** (**Rx**.Event_X.Type = Cancel) **AND** (**Rx**.Event_X.Category = Input) **AND** (**Rx**.Event_X.Action = **Ry**.Event_Y.Action) **AND** (**Rx**.Event_X.Object = **Ry**.Event_Y.Object) **AND** (**Rx**.Event_X.Object.Pre-State = **Ry**.Event_Y.Object.Pre-State) **}** **THEN Rx Cancel Ry**

4 DISCOVERING INTERACTIONS

The proposed process is divided into stages. The division in stages allows an efficient mechanism for reviewing the results of each of the stages and to negotiate the progress of the most important stage of the process (Interactions Detection).

The proposed approach consists of six main steps organized in a specific order to facilitate the interactions detection. In each step, different tables and graphs are produced with the purpose of facilitating and increasing the precision of the process. In the main step (step 4), an analyst identifies the different interaction types based on interactions detection rules defined in Section 3.1. The six steps are briefly described below:

Step 1. Listing the Requirements: Initially the requirements are listed and described textually using natural language. The requirements are ordered according to their complexity, and the complex requirements are decomposed into simpler others.

Step 2. Extracting Requirements Attributes: A requirements analyst begins the process of identification of attributes for each one of the requirements (events, action, objects, agents and resources). Table 1 describes the list of attributes.

Step 3. Associating Requirements and Events: After the requirements were decomposed in their attributes, it begins the process of association of each one of the requirements to its related events. Then, the attributes of the events must be identified (Table 2).
To facilitate and avoid unnecessary comparisons in the detection step, the identification of the common events is done. This is necessary because the requirements interact mostly through events, besides having common actions, resources and objects.

Step 4. Detecting Interactions: The set of interaction detection rules described in Section 3.1 is applied on these requirements.

Step 5. Validating Interactions: The interactions identified on step 4 must be validated. A negotiation with the users, in which the conflict or inconsistencies interactions must be resolved, is also part of this step.

Step 6. Specifying Requirements: The requirements are documented with additional information (requirements and their interactions). This documentation will be very important in the next stages of the software development process.

5 CASE STUDY

We have been evaluating the effectiveness of our method in several domains (Sarmiento, 2008). To illustrate the method we show a case study using the Lift Control System. In this case study, a set of 14 requirements (Table 7) describes the basic operation of a simple lift. A detailed and complete description of this case study can be found in (Shehata, 2005).

The Lift is composed of:

- A Call Button in each floor.
- An Open-Door Button inside the Lift.
- Buttons for each floor inside the lift.

All the results produced and a more detailed description of each one of the method steps is presented in (Sarmiento, 2008).

Step 1. Listing the Requirements: The requirements list is reproduced in Table 7.

Step 2. Requirements Attributes: After the requirements are listed, their attributes are identified (Table 8).

Step 3. Associating Requirements and Events: Initially, the process of Identifying Common Events is done because the requirements interact principally through events (See Table 9).

Then, the requirements are associated with their events, besides specifying each one of these events (Category, Type, etc.). Tables 10 and 11 show the attributes of events E5 and E13 listed in Table 9.

Step 4. Detecting Interactions: The method then proceeds to the identification of the interactions. Tables 12 and 13 show a subset of the identified interactions, applying the interaction detection rules described in Section 3.1.

Step 5. Validating Interactions: Table 14 shows a summary of the interactions identified applying the method.

Step 6. Specifying Requirements: The different tables produced in each one of the method steps compose the requirement specification document.

Table 7: The Lift Control System Requirements.

IDR	Description
R1	The lift is called by pressing a call button, either at a floor or inside the lift.
R2	Pressing a call button is possible at any time.
R3	When the lift passes by floor K, and there is a call for this floor, then the lift will stop at floor K.
R4	When the lift has stopped, it will open the doors.
R5	When the lift doors have been opened, they will close automatically after d time-units.
R6	The lift only changes its direction when there are no more calls in current direction.
R7	When there are no more calls, the lift stays at the floor last served.
R8	As long as there are unserved calls, the lift will serve these calls.
R9	When the lift is halted at floor K with the doors opened, a call from floor K is not taken into account.
R10	When the lift is halted at floor K with door closed and receives a call from floor K, it reopens its doors.
R11	Whenever the lift moves, the doors must be closed.
R12	The closing of a door may be prevented by pressing the open-door button.
R13	When something blocks the door, the lift interrupts the process of closing the door and reopens the doors.
R14	When the lift is overloaded, the door will not close.

Table 8: Identifying Requirements Attributes.

ID	Event	Action	Object	Resource
R1	Pressing the call button. The lift is called from the floor K.	Call the lift.	The lift. The Call Button (in/out).	
R2	Pressing the call button.		The Call Button (in/out).	
R3	The lift passing through the floor K. The lift is called from the floor K. The lift is stopped.	Stop the lift in the floor K.	The lift. The floor.	
R4	The lift is stopped. The doors are opened.	Open the doors.	The lift. The doors.	
R5	The doors are opened. The doors are closed.	Close the Doors automatically after d time units.	The lift. The doors. The time counter.	
R6	There are no more calls in the current direction.	Change the lift direction is possible.	The lift. A call to lift	
R7	There are no more calls. The lift is stopped.	The lift stays at floor last served.	The lift. A call to lift	
R8	There are calls.	The lift serves the calls.	The lift. A call to lift	
R9	The lift is called from the floor K. The lift is stopped. The doors are opened.	Call the lift.	The lift. The doors. The floor.	
R10	The lift is called from the floor K. The lift is stopped. The doors are closed. The doors are opened.	Open the doors.	The lift. The doors. The floor.	
R11	The lift is moving.	Close the doors.	The lift. The doors.	
R12	Pressing the open-door button. The doors are opened.	Close the Doors automatically after d time units.	The lift. The doors. The open-door button.	
R13	Something blocks the doors. The doors are opened.	Close the Doors automatically after d time units.	The lift. The doors.	The block sensor.
R14	The lift is overloaded. The doors are opened.	Close the Doors automatically after d time units.	The lift. The doors.	The Overload sensor.

Table 9: Identifying Events.

IDE	Description	Related Requirements
E1	Pressing the call button.	R1,R2
E2	The lift is called from the floor K.	R1, R3, R9, R10
E3	The lift passing through the floor K.	R3
E4	The lift is stopped.	R3, R4, R7, R9, R10
E5	The doors are opened.	R4, R5, R9, R10, R12, R13, R14
E6	The doors are closed.	R5, R10
E7	There are no more calls in the current direction.	R6
E8	There are no more calls.	R7
E9	There are calls.	R8
E10	The lift is moving.	R11
E11	Pressing the open-door button.	R12
E12	Something blocks the doors.	R13
E13	The lift is overloaded.	R14

Table 10: Identifying Attributes for Event E5.

Attribute	Description
IDE	E5
IDR	R5
Type	Link
Category	Input
Action	Close the Doors automatically after d time units.
Object	Pre-state: The Doors are opened. Next-state: The Doors are closed.

Table 11: Identifying Attributes for Event E13.

Attribute	Description
IDE	E13
IDR	R14
Type	Cancel
Category	Input
Action	Close the Doors automatically after d time units.
Object	Pre-state: The Doors are opened. Next-state: The Doors are opened.

Table 12: Requirements Interaction: R3 -> R1.

Interaction Type	Require. See Table 5 *(require)*.
Requirements	R3 -> R1
Explanation	To stop the lift at floor K, it must be called pressing the button. The event produced in the requirement R1 (E2) stimulates the execution of the action of R3.

Table 13: Requirements Interaction: R13 -> R5.

Interaction Type	Cancel. See Table 6 *(cancel)*.
Requirements	R13 -> R5
Explanation	Closing the doors is canceled when something blocks the door. The event of the requirement R13 (E12) cancels the execution of the action of R5.

Table 14: The Identified Interactions.

Requirement	Interacting Requirement	Interaction Type
R8	R9	cancel
R9	R1	cancel
R12	R5	cancel
R13	R5	cancel
R14	R5	cancel
R12	R8	Negative Impact
R12	R8	Negative Impact
R12	R8	Negative Impact
R3	R1	require
R4	R3	require

6 CONCLUSIONS AND RESULTS

In this paper, we have presented our approach to specify, detect, and automatically discover the interactions among software requirements. The approach is based on events and actions and uses a semi-formal method. The method requires neither the user intervention nor any external knowledge for the *identification of interactions*. The method is able to identify most interaction types described in the literature (negative and positive interactions).

The proposed approach detected all interactions reported by Heisel and Souquières (2001) and Shehata (2005), besides extra interactions not detected by other approaches (mostly positive interactions). These are the interaction types that allow the behaviour modelling of the system, mainly those that enable interaction between objects. Table 15 summarises this comparison.

Table 15: Comparing Results.

Approach	Positive Interactions	Negative Interactions
Proposed Approach	13	8
(Heisel and Souquières, 2001)	-	6
(Shehata, 2005)	-	7

The method reduces the human participation, because the process of interactions detection (applying interaction detection rules) can be automated and implemented through a computational support tool.

To facilitate and avoid unnecessary comparisons during the interactions detection, the process of identification of the common events and actions associated to the requirements was conceived.

The knowledge of interactions allows resolving eventual conflicts and planning the requirements implementation (priorities) considering dependencies interactions.

The method proposed in this paper works well if a requirements analyst can properly identify requirements attributes as shown in Table 7. However, this work is less complex than to specify the requirements using a formal specification language.

We plan to expand the method to cover additional interactions types, which were not considered in this paper. Moreover, we intend to work on the improvement of the method by identifying and complementing the rules for interactions detection. This will enable us to make the algorithms more efficient and precise.

We are also working on a software prototype to support the user to validate, improve or reject the interactions identified by our method.

ACKNOWLEDGEMENTS

This work was partially funded by CNPq grants 311454/2006-2 and 305900/2005-6.

REFERENCES

Dahlstedt, Å.G., Persson, A., 2003. Requirements Interdependencies - Moulding the State of Research into a Reseach Agenda. In *Proceeding of the Ninth International Workshop on Requirements Engineering: Foundation for Software Quality*, Austria, 2003.

Robinson, W. N., Pawlowski, S. D., Volkov, V., 2003 Requirements interaction management. *In ACM Computing Surveys 35 (2)*, 2003, pages 132-190.

Shehata, M., 2005. Detecting Requirements Interactions using Semi-Formal Methods. PhD. Thesis (Doctor of Philosophy), Department of Electrical and Computer Engineering, Calgary University, Canada, 2005.

Yuqin, L., Chuanyao, Y., Chongxiang, Z., Wenyun, Z., 2006. An Approach to Managing Feature Dependencies for Product Releasing in Software Product Lines, *In Proc. of the International Conference on Software Reuse, LNCS 4039*, 2006, pages 127-141.

Zhang, W., Mei, H., Zhao, H., 2006. Feature-driven Requirement Dependency Analysis and High-level Software Design, *Requirements Engineering Journal*, volume 11, number 3, 2006, pages 205-220.

El-Ansary, A., 2002. Behavioral Pattern Analysis: towards a new representation of systems requirements based on actions and events. In The ACM Symposium on Applied Computing, Madrid, Spain. 2002.

BPMN, 2006. Business Process Modeling Notation, OMG.

Calder, M., Magill, E., 2000. Feature Interactions in Telecommunications and Software Systems VI, Amsterdam: IOS press, 2000.

Lamsweerde, A. V, Darimont, R., Letier, E., 1998. Managing Conflicts in Goal-Driven Requirements Engineering. In IEEE Trans. on Software Engineering, volume 24, number 11, 1998, pages 908-926.

Pohl, K., 1996. Process-Centered Requirements Engineering, John Wiley & Sons Inc.

Mylopoulos, J., Chung, L., Yu, E. 1999. From object-oriented to goal-oriented requirements analysis. Commun. ACM, 42, pages 31–37.

Heisel M., Souquières, J., 2001. A heuristic algorithm to detect feature interactions in requirements. In Language Constructs for Describing Features, M. Ryan, Ed.: Springer-Verlag London Ltd, 2001.

Sarmiento, E.C., 2008. Detecting interactions between requirements in software development. M.Sc. Dissertation in preparation, Graduate Program in Informatics, Federal University of Rio de Janeiro, Brazil (In Portuguese).