

IRIS: A Semi-Formal Approach for Detecting Requirements Interactions

Mohamed Shehata¹

Armin Eberlein²

Abraham Fapojuwo¹

¹ Dept. of Electrical & Computer Engineering, University of Calgary, 2500
University Drive NW, Calgary, AB, Canada

² Dept. of Computer Engineering, American University of Sharjah, PO Box 26666,
Sharjah, UAE
{Shehata; Eberlein; Fapojuwo}@enel.ucalgary.ca

Abstract

Requirements engineering is considered a critical phase of the software development life cycle. However, because of the complexity of today's projects, requirements often have a negative impact on each other. Requirements interaction detection is an important activity for the discovery of such unwanted interactions. Commonly used detection processes are oriented towards the telecommunication domain and are done using either human experts or formal approaches. This paper presents IRIS, which stands for Identifying Requirements Interactions using Semi-formal methods. The novelty of IRIS is threefold: First, IRIS uses semi-formal methods for the detection of interactions between requirements. This helps to fill in the gap between the commonly used informal and formal approaches. Secondly, IRIS is a domain independent approach, which means that it is not limited to the telecommunications domain but can be used in any field. Thirdly, IRIS has a basic core as well as extension hooks for future expansion through the creation of new plug-ins that can be attached to the hooks. This paper first presents an overview of IRIS along with its basic core. It then describes the customizability of IRIS through hooks and plug-ins. Finally it presents the customization of IRIS using different plug-ins for different domains as well as a summary of the results obtained from these domains.

Keywords: Requirements engineering, Requirements interactions, IRIS, Semi-formal approaches.

1. Introduction

The successful development of software systems requires an unambiguous, complete, and non-conflicting set of requirements. Requirements-related problems have a major effect on the delivery of a software system on time and within budget [1]. However, because of the nature of software development, numerous problems hinder the development of high-quality requirements. One of the requirements-related problems that can lead to

costly repairs if not found early is requirements conflicts or so-called requirements interactions. Requirements interaction is the situation when two or more requirements have an effect on each other. Requirements interactions can be caused by the following [2], [3]:

1. Requirements are often collected from different stakeholders with different perspectives and views. This can lead to interacting requirements between the different stakeholders.
2. Many projects employ reuse in order to reduce the project's development time and cost. However, the reused requirements can have negative effects on the new set of requirements for the project.
3. Certain software development methods such as component-based development, product lines, product families, require the collection of new requirements to be integrated into the already existing base system. These new requirements can have a negative impact on existing requirements. Moreover, when assigning values to parameterized reusable requirements, these values might cause interactions. For instance, consider a requirement stating that the response time of the system should not exceed X ms and another parameterized requirement that the system shall use Y processor. If parameter X is assigned a value of 0.5 ms and Y is an 8088 processor, these two requirements might be in conflict. However, if X is 0.5 ms and Y a Pentium IV processor, both requirements might be able to be met at the same time. This shows that requirements conflicts heavily depend on the values of parameters.

This paper presents a semi-formal approach to address the problem of requirements interaction detection. IRIS, which stands for Identifying Requirements Interactions using Semi-formal methods, uses semi-formal methods for the detection of interactions between requirements

2. Related Work

One of the commonly used methodologies in developing software systems is feature-based development. However, this approach has been hampered

by the so-called feature interaction problem. A feature is defined as an identifiable piece of functionality that is added on top of a base system to extend it. Feature interaction is the situation where two features contradict or have a negative impact on each other. The problem of feature interactions has been well researched in the telecommunications domain. Calder *et. al.* give a very extensive review of the different feature interaction methodologies and research efforts in [4]. However, some observations were noticed by the authors of this paper. For example, most of these research efforts are directed towards the telecommunications domain (e.g., [5] [6] [7]). Although some methodologies claim to be domain independent (e.g. [8]), they have never been applied to a different domain. Furthermore, most methodologies use formal methods for detecting interactions. Although, formal methods (if used correctly) are very accurate in detecting interactions, they are very time consuming and expensive. Other approaches employ human domain experts to detect interactions. But such experts are expensive and prone to human errors [9].

In the past the term feature interaction detection was very limited in scope and usually only referred to detecting interactions between *features* of the *telecommunications domain* using *formal methods*. Requirements interaction management was introduced in [10] as “the set of activities directed towards the discovery, management, and disposition of critical relationships among a set of requirements”. Requirements interaction is very similar to feature interaction as both of them try to identify the relationships between features or requirements. However, requirements interaction has a broader scope than feature interaction:

1. Requirements interaction is more general than feature interaction as it considers functional as well as non-functional requirements.
2. Requirements interaction considers requirements inconsistencies.
3. The term feature interaction is generally perceived to be limited to the telecommunication domain. But requirements interaction is a phenomenon that can occur in any domain.
4. While feature interaction tends to consider only technical interactions between requirements, requirements interaction additionally considers relationships between requirements caused by the heterogeneity of stakeholders

Our research addresses the wider area of requirements interactions. This paper first presents an overview of IRIS in section 3 (IRIS basic core and IRIS customizability concept). In section 4, three case studies are presented that show how IRIS has been customized and used with different plug-ins in different domains. Finally, section 5 ends the paper with conclusions and our intended future work.

3. Introduction to IRIS

IRIS is a semi-formal approach that can be applied to various domains. It consists of a basic core that contains the main components and steps that must be performed at all times. If a more thorough interaction analysis for a specific domain is required, plug-ins can be added to specific hooks in the basic core to increase its effectiveness. The users can develop their own plug-ins as long as it complies with the design rules described later in this paper. In the following subsections the basic core and the different plug-ins developed so far are presented.

3.1. The Basic Core of IRIS

The basic core of IRIS consists of the main steps, tables, graphs, and interaction detection guidelines that are applied regardless of the domain or the type of system under development. In addition to these, the basic core contains predefined hooks where domain-specific plug-ins can be added in order to extend the basic core. IRIS basic core is a systematic approach based on several steps in a specific order to facilitate the detection of requirements interactions. Different tables and graphs are developed and in a final step the analyst reviews these tables and graphs using a set of requirements interaction detection guidelines to detect interacting requirements. IRIS basic core is graphically presented in Figure 1.

The IRIS basic core methodology consists of six main steps. The requirements are first written down in a textual requirements document in natural language. Using six steps, the requirements are gradually translated into a graphical and tabular representation. The following briefly describes the steps. A full description of them can be found in [11], [12].

- Step 1: Organizing requirements: In this step all system requirements are classified into one of the following two categories:
 - Requirements describing system axioms (A system axiom is a property needs to preserve at all times).
 - Requirements describing system dynamic behavior (A dynamic behavior requirement is a requirements that describes how the system should behave in terms of states change and actions when triggered by a trigger event).
This step is shown in Figure 1 as the classification of the requirements into one of the two categories: system axioms and dynamic behavior.
- Step 2: Filling in requirements tables: Two tables are generated in this step:
 - The first table is used to represent requirements describing system axioms using the attributes *Rules* and *Conditions* (more attributes can be added when needed)

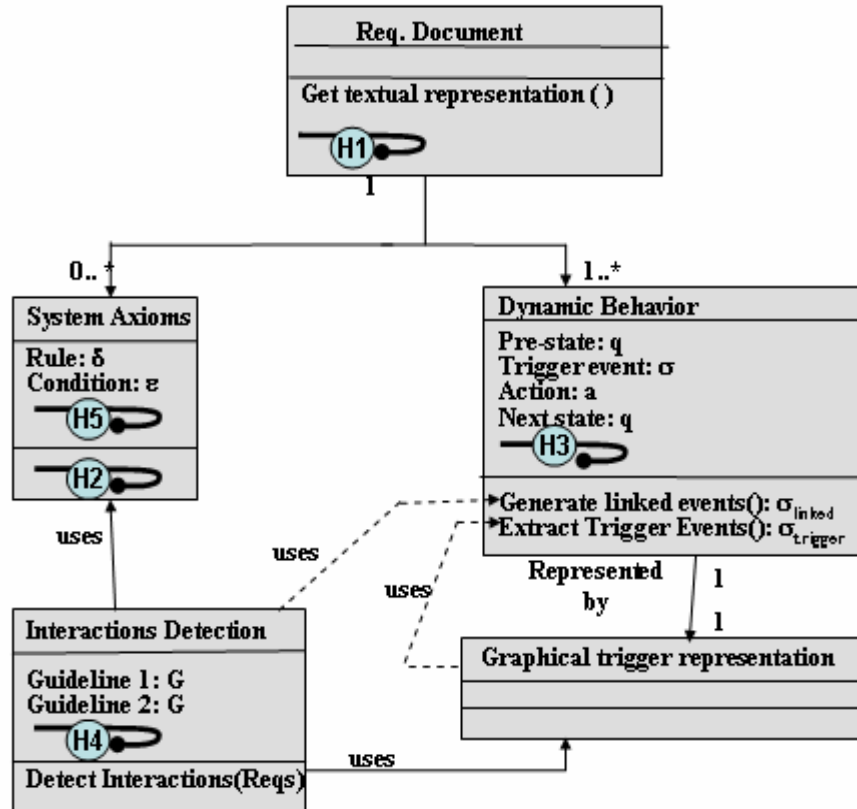


Figure 1: Basic core of IRIS

- The second table is used to represent requirements describing system dynamic behavior using the four attributes *Pre-state*, *Trigger event*, *Action*, and *Next state* (more attributes can be added when needed).

This step is done by filling in the values of the different attributes of the system axioms and dynamic behavior classes shown in Figure 1.

- Step 3: Extracting trigger events and tracing them to their requirements: This step identifies and extracts all the different trigger events from the requirements that describe dynamic system behavior. This step is shown as the process *Generate Trigger Events()* in the dynamic behavior class in Figure 1.
- Step 4: Identifying linked events: During this step a table is developed that contains all linked trigger events. A trigger event is called a linked trigger event if it can lead to other trigger events. This step is shown as the process *Generate Linked Events()* in the dynamic behavior class in Figure 1.
- Step 5: Representing each event-triggered requirement in a graphical form: In this step, a graphical notation is used to link each trigger event with the requirements that it triggers. This graphical representation will later facilitate the detection of interactions between the requirements. This step is represented by the class *Graphical trigger representation* in Figure 1.

- Step 6: Detecting interactions: In this step, the analyst detects interactions between requirements using specific interactions guidelines. This step is shown in Figure 1 by the class *Interaction Detection*.

The guidelines in Step 6 were derived from a general interaction taxonomy that describes all the possible categories and scenarios of requirements interactions [13]. The guidelines are a description of when two requirements are considered to be interacting. The basic core of IRIS contains the following two guidelines:

- Guideline 1: Two requirements interact if they are triggered by the same trigger event and both of them have the same pre-state but have contradicting actions or contradicting next states.
- Guideline 2: Two requirements interact if they are triggered by two linked events but have contradicting actions or contradicting next states.

The hooks which are represented by H1, H2, H3, H4 and H5 in Figure 1 are insertion points for plug-ins in order to extend the basic core. Each hook has a unique name that starts with an H followed by a number to identify this hook. The type of the possible plug-ins can vary from just adding an attribute to a table to adding a complex step. However, not every type of plug-in can be inserted at every hook. For example a plug-in that is

inserted at hook H3 must be an attribute, while a plug-in that is inserted at hook H4 must be a guideline. The different plug-ins that have been developed so far are described in the next sub-section.

3.2. IRIS Plug-ins

A plug-in is something that is added to IRIS basic core to extend its capability of detecting interactions. Plug-ins have a certain format that has to be followed when new ones are designed. A plug-in has three main parts: The first part identifies the type of the plug-in. The second part is the plug-in main body. The third part identifies where this plug-in can be inserted in the basic core of IRIS. Figure 2 shows a description of the construction of a plug-in.

The Type is a four letter abbreviation that describes the type of the plug-in. A plug-in can have one of the following types:

- **Attribute (ATTR):** An attribute is a description of a specific part of a requirement. For example, every dynamic requirement is described by four attributes (pre-state, trigger, action, and next state). These attributes correspond to columns in the dynamic behavior table that is generated during step 2 of the basic core. In new domains, new attributes may be needed so that requirements can be fully described. For example, in the smart homes domain (see case study 3 in section 3.3), many requirements have parameters in their body and therefore two new attributes are needed. The first one, called *Variables*, contains the list of variables that are used in the requirement as well as their data types. The second attribute is called *Variable Range* and can be used to restrict the data range that can be assigned to a variable.
- **Guideline (GDLN):** A guideline is a description of the situations in which two requirements interact. These guidelines are derived from a requirements interaction taxonomy [13] that describes the different categories and scenarios of requirements interactions. The number

of guidelines used determines the reliability of detecting design or implementation problems.

- **Table (TABL):** A table represents requirements or events using a set of attributes. Each attribute corresponds to a column in the table. A plug-in can be a table that extends the basic core of IRIS to be able to represent certain aspects as needed. For example, when representing the system axioms in a system, some of these system axioms might be non-functional requirements (quality requirements). It is sometimes desired to represent different development strategies of these non-functional requirements as suggested by the stakeholders. In order to do so, a new table plug-in can be inserted at hook H2 to capture these different development strategies along with their pros and cons.
- **Step (STEP):** A step is an independent component that can generate its own set of tables and graphs. This type of plug-in is needed when there is a certain step that is not necessarily applied at all times, like representing the requirements in a graphical notation to make sure that the analyst fully understands how each dynamic requirement behaves in terms of states and actions. This type of plug-in is also needed in a possible preparation phase before applying the basic core of IRIS.

The P in a circle (Figure 2) describes where this plug-in can be *Placed* and inserted. For instance, if a plug-in can be inserted into hook H2, the P in the plug-in is assigned the value H2. This prevents any mis-location of plug-ins. Finally, the plug-in body contains the main part of the plug-in and describes what this plug-in is, when and how the analyst should use it. The plug-in body has the following parts:

- **What:** States what this plug-in is
 - **Name:** A unique descriptive name of the plug-in
 - **Description:** A textual description of what this plug-in is
 - **Construction:** The internal construction of the plug-in

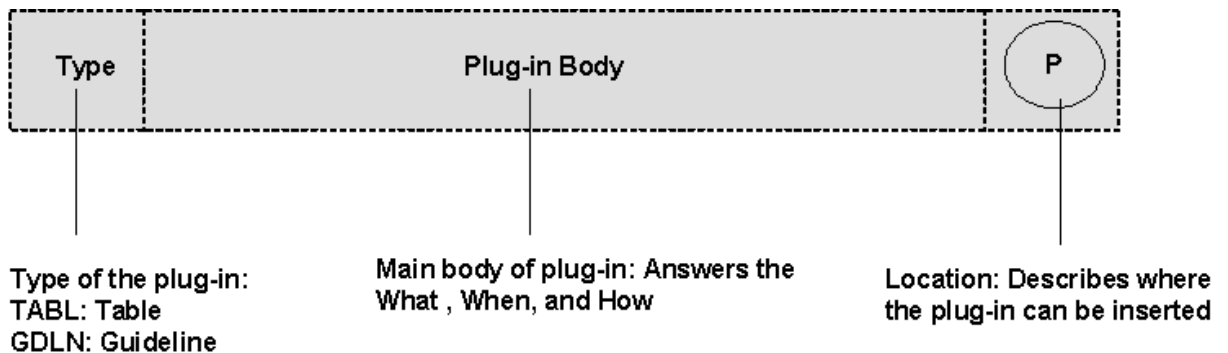


Figure 2: Format of a plug-in

- **When:** States when to apply this plug-in
 - **Problems it overcomes:** A description of what types of problems this plug-in can overcome when used
 - **Expected enhancement:** A description of the expected enhancement this plug-in will bring when used
 - **Example of application:** A sample description of when to use the plug-in
- **How:** States how to apply this plug-in
 - **Instructions:** A set of instructions on how to insert and use this plug-in plus any other instructions.

So far we have designed twelve plug-ins that have the structure described above and are fully documented.

Table 1 presents a full description of the plug-in named “Graphical representation of individual requirements” as an example. The other plug-ins are only briefly described below.

- *Resources:* This is an attribute plug-in. It corresponds to a new column in the system axiom table or the dynamic behavior table. The new column describes what resources each requirement needs in order to be fully met. This plug-in is connected to hook H3 or connected to hook H5.
- *Parameter assignment:* This is a step plug-in. It is used to find any parameterized parts in the given set of requirements. Then these parameterized parts are replaced by parameters (e.g., X, Y ...etc). For example, consider a requirement that has a part stating “the lights will switch on in a certain place”. Certain place is a parameterized part and the Parameter assignment plug-in replaces this part with a parameter X. The requirement now should be “the lights will switch on in X”. This plug-in is connected to hook H1.
- *Variables attribute:* This is an attribute plug-in. It corresponds to a new column in the system axiom tables or the dynamic behavior tables. The new column describes the different variables used in each requirement along with the data type allowed for this variable. This plug-in must be used in conjunction with the “Parameter assignment” plug-in. It is connected to hook H3 or hook H5.
- *Variables Range attribute:* This is an attribute plug-in that has to be used in conjunction with the *Variables attribute* plug-in. It corresponds to a new column in the system axiom tables or the dynamic behavior tables. The new column describes the allowed range of values that can be entered for each variable. This plug-in is connected to hook H3 or connected to hook H5.
- *Functionalities identification:* This is a step plug-in and is used when a single requirement is complex and describes different functionalities. For example a

requirement for an intruder alarm has many sub-requirements and functionalities. The goal is to simplify the parent requirement and to separate the different encapsulated functionalities. This plug-in is connected to hook H1.

- *Graphical representation of individual requirements:* This is a step plug-in that is connected to hook H1. It corresponds to a complete step that is performed before step 2 in the basic core. This plug-in is used when the given set of requirements are complex and therefore must be fully understood before proceeding with the rest of the interaction detection approach steps. A complete description of this plug-in is given below in Table 1.
- *System axioms strategies:* This is a table plug-in, i.e., a new sub-table is inserted as part of the parent system axioms table. This table is completed during step 2 of the IRIS basic core steps and describes what strategies are necessary to implement the system axioms. This plug-in is connected to hook H2.
- *Guideline 3:* This is a guideline plug-in. It is used to detect interactions between two requirements if both of them are system axioms and the rule part of one requirement contradicts the rule part of the second requirement. This plug-in is connected to hook H4.
- *Guideline 4:* This is a guideline plug-in. It is used to detect interactions between two requirements if one of them is a system axiom and the other one describes dynamic behavior and the action part of the dynamic behavior requirement is in conflict with the rule part of the system axiom. This plug-in is connected to hook H4.
- *Guideline 5:* This is a guideline plug-in. It detects interactions between two requirements that are triggered by the same trigger event, have different pre-states, and the activation of one requirement bypasses the activation of the second requirement. This plug-in is connected to hook H4.
- *Guideline 6:* This is a guideline plug-in. It can be used to detect interactions between two requirements that are triggered by linked events, have different pre-states, and the activation of one requirement bypasses the activation of the second requirement. This plug-in is inserted at hook H4.
- *Guideline 7:* This is a guideline plug-in. It can detect interactions between requirements R1 and R2 in the following situation: Requirements R1 and R2 are triggered by linked events (two events are said to be linked when the occurrence of the first event is logically followed by the occurrence of the second event). R1 is already triggered and executing an action. Then R2 is activated and requires an action that will cancel the action of R1 before it is completed. This plug-in is connected to hook H4.

Table 1: Details of the plug-in *Graphical representation of individual requirements*

| | | | | |
|--------------|-------------|--|--|--|
| Type: | STEP | | | |
| Body: | What | Name | Graphical representation of individual requirements | |
| | | Description | A complete step that is carried out to graphically represent each individual requirement that describes dynamic behavior of the system. This is to ensure that the analyst fully understands the behavior of the requirements. | |
| | | Construction | The execution of this step requires the following activities: 1. Select every requirement that describes dynamic system behavior, list it separately, and read it carefully. 2. Identify a suitable graphical representation (e.g., State charts [14], CRESS [15], UCM [16]). 3. Represent each of the selected requirements graphically using the chosen graphical notation. 4. If it is difficult to represent the requirement, the analyst needs to restate the requirement and possibly consult with the source/stakeholder of the requirement in order to better understand it. 5. Go back to activity 3 until all requirements have been addressed. | |
| | When | Problems this plug-in overcomes | 1. Complexity of requirements 2. Ambiguity of requirements 3. Lack of understanding of requirements 3. Clarification of wrong assumptions or wrong judgments | |
| | | Expected enhancements | 1. Reduced requirements ambiguity 2. Reduced difficulty filling in the requirements tables in step 2 of the basic core of IRIS 3. Improved accuracy of the requirements attributes (e.g. pre-state, trigger event, action, next state) 4. Improved interaction detection and prevention of false interactions | |
| | | Sample of application | This step has been applied in a case study to identify interactions between the requirements of a lift system. Refer to [2] for an example application. | |
| | How | Instructions | 1. This step is directly applied after step 1 of IRIS basic core. 2. This step is applied only to requirements describing dynamic system behavior. 3. Any graphical notation can be used that is able to fully represent the four basic attributes: pre-state, trigger event, action, and next state. | |
| | Ⓟ | H1 | | |

4. Customizing IRIS for Different Domains

As stated earlier, IRIS is a semi-formal approach that can be customized for different domains. Customization is achieved by extending the basic core of IRIS using plug-ins that are added to so-called hooks. This extensibility gives the approach great flexibility and adaptability to the needs of various domains.

In this section we show how these plug-ins have been used with the basic core for different domains. So far, we have conducted three case studies from three different domains:

1. Control domain: A set of requirements that describe the operation of the lift system was analyzed.
2. Telecommunications domain: A set of requirements that describe different telephony features (e.g., CFBL, CW, TWC, TCS...etc) [17] was analyzed.
3. Smart homes domain: A set of user policies and features that describes the operation of a smart home was analyzed.

Although smart homes domain seems to be similar to the control domain, it belongs to a separate category because smart homes are concerned with user policies which is different from the traditional control domains. In the following subsections, a brief description of each case study, a description of the plug-ins used, and a summary of the results obtained are given. Also, a comparison of the obtained results with other results reported in the literature is given. The following points are worth considering when reviewing and comparing our results with those reported in the literature:

- The presented results were obtained using IRIS which is a semi-formal approach. This means that no costly formal models or expensive human experts were used.
- The detected interactions are of two types: Interactions detected using the guidelines of the basic core and interactions detected using the additional plug-ins. The guidelines of the basic core capture all critical interactions (e.g., non-determinism) in any system. The use of more detection guidelines (by inserting plug-ins) helps to detect more interactions.

The number of detected interactions versus the overall number of actual interactions is something that is up to the customer (e.g. in low criticality systems, the customer might want to detect only critical interactions). The need to detect more interactions or to capture all interactions means the use of more plug-ins and therefore more cost and time.

4.1. Case Study 1: The Lift Control System

The lift control system can be considered as part of the control domain. In this case study a set of 14 requirements describing the behavior of a lift system was analyzed. A

more detailed description of the case study can be found in [11].

4.1.1. Plug-ins Used in the Lift Control System

We first list the problems encountered in this case study and then describe which plug-ins were used to overcome these problems.

1. In the beginning some requirements were ambiguous. The plug-in “*Graphical representation of individual requirements*” was used to graphically represent requirements and understand their exact behavior. This helped resolve the ambiguities that existed earlier on. The reformulation of the textual descriptions of some requirements as part of the plug-in greatly clarified the requirements and improved their understanding.
2. Since the set of requirements used in the case study contained requirements describing system axioms, interactions between these system axioms had to be analyzed. The plug-in “*Guideline 3*” was used to detect interactions that can arise between system axioms. The plug-in provided instructions on how to detect such interactions.
3. Since the set of requirements used in the case study contained requirements describing system axioms as well as dynamic behavior, the analyst had to make sure that no dynamic behavior requirements violate or interact with any system axioms. The plug-in “*Guideline 4*” provided instructions on how to detect such interactions.

4.1.2. Summary of the Obtained Results

A summary of the results obtained as well as a comparison with results reported by Heisel et al [18], [19] are presented in Table 2. Three plug-ins were used in this case study.

Table 2: Summary of the lift case study results

| | Simple Interactions | Indirect interactions (sequential interactions by linked events) | Missed interactions |
|----------------------|---------------------|--|---------------------|
| Heisel et al [8] [9] | 5 | 0 | 2 |
| IRIS | 5 | 1 | 1 |

4.2 Case Study 2: The Telecommunications Features

The case study was conducted on a set of eight telecommunication features that were provided by the feature interaction contest held in 2000 [17]. The textual descriptions of the features were at a suitable level of abstraction, i.e., they did not contain implementation details. Such low-level details are currently still beyond the scope of IRIS. More details on the case study can be found in [20].

4.2.1. Plug-ins Used in the Telecommunications Case Study

The descriptions of the features provided by the contest organizers [17] were very detailed and addressed all the questions that might be asked. The authors therefore only had to use the following two guideline plug-ins that helped detect interactions related to high-level design problems:

1. The plug-in “*Guideline 5*” was used to detect interactions caused by two features being triggered by the same trigger event and one feature bypassing the trigger of the other feature.
2. The plug-in “*Guideline 6*” was used to detect interactions caused by two features being triggered by linked trigger events and one feature bypassing the trigger of the other feature.

4.2.2 Summary of the Obtained Results

A summary of the results obtained is presented in Table 3 together with results reported by other researchers. The number of plug-ins used in this case study was two. It must be noted that all the other contestants (P, N, and H) used formal methods which included building formal models of the system and validating these models. Formal methods are quite accurate in detecting interactions but they are very time consuming and costly. On the other hand IRIS uses semi-formal methods, i.e., requires less time and money because it reduces the number of comparisons that an expert would have to do using pairwise comparisons. Therefore, assuming equal cost and time for comparing two requirements, IRIS does achieve good results using less time and less cost. When comparing IRIS with other approaches using formal methods, it achieved very good results at lower cost and in less time.

Table 3: Summary of the obtained results for the telecommunications feature case study

| Approach | Detected | Missed |
|----------------------|----------|--------|
| IRIS | 21 | 2 |
| Plath & Ryan [21] | 22 | 1 |
| Nakamura et al. [21] | 21 | 2 |
| Hall [21] | 22 | 1 |
| Samborski [21] | 8 | 15 |

4.3. Case Study 3: Smart Homes Policies

This case study identifies interactions between user policies within a smart home [22]. A smart home user has a set of user-defined policies. The following is an example of a policy: At sunset, the home master control shall close the windows, close the curtains, and turn on the lights in the living room. Each of the policies tends to involve different features such as the windows control

feature, the curtains control feature, and the lights control feature. Each feature has many functionalities that it can perform. For instance, the light control feature can turn on the lights, turn off the lights, dim the lights to a certain level, use timers and special triggers to do a combination of these functionalities.

A policy consists of several functionalities that perform the required actions. For example, in the above mentioned user policy, the policy involves the functionality turn on light when a certain trigger is received. This means that functionalities link policies to features. Therefore, the case study focused on detecting interactions among functionalities. The 16 features could be broken down into 35 functionalities which were then analyzed for interactions. More details of the case study can be found in [13].

4.3.1. Plug-ins Used in the Smart Homes Policies

The domain of smart homes is relatively new. It contains numerous features, requirements, and physical network elements the functions of which are determined by user policies. The system is reasonably complex and distributed, so several plug-ins were needed to customize IRIS for this case study. The following plug-ins were used:

1. The different functionalities in the case study had many parameters, i.e., the plug-in “*Parameter assignment*” was used to replace the parameters with variables.
2. These variables (e.g. X, Y, etc) must be included into the tables and graphs created within the basic core of IRIS. In order to do this, the plug-in “*Variables attribute*” was used to add new columns to the tables.
3. The plug-in “*Variables Range*” must be used in conjunction with the previous plug-in in order to indicate the allowed range of values that each variable can have. The values assigned to each variable have a major influence on possible interactions between functionalities.
4. In order to link the features of the smart home to user policies and functionalities, the plug-in “*Functionalities identification*” was used.
4. Since the case study had requirements describing system axioms, it was necessary to look for interactions between these system axioms. The plug-in “*Guideline 3*” was used for this purpose.
5. Since the case study contained requirements describing system axioms it was necessary to ensure that no dynamic behavior requirement violates or interacts with any system axiom. The plug-in “*Guideline 4*” was used to detect such interactions.
6. This case study is about a distributed system and each functionality requires a considerable amount of time to complete (e.g., when the temperature drops, it takes time for the heating control feature to adjust the

temperature again). During this time, a new functionality might be triggered that requires an action that contradicts the first operation. Therefore the plug-in “Guideline 7” was used to detect interactions due to timing problems. Such interactions can be termed “sequential interactions”.

4.3.2. Summary of the Obtained Results

Table 4 summarizes the interactions detected using IRIS. To our knowledge, no results of other researchers on smart homes are reported in the literature, therefore a comparison of IRIS with other results is not possible.

Table 4: Summary of the obtained results for the smart home policies case study

| | |
|--------------------------------|----|
| Number of Features | 16 |
| Number of functionalities | 35 |
| Number of interaction detected | 88 |
| Number of plug-ins used | 6 |

5. Conclusions and Future Work

This paper presents the importance of using semi-formal methods in detecting requirements interactions. The paper presents IRIS, a semi-formal approach for detecting requirements interactions. An important aspect of IRIS is its customizability to different domains. In order to do this, the basic core of the approach can be extended using different plug-ins. Guidelines for when, where, how, and why each plug-in should be used were developed. The approach was validated by applying it to three case studies from three different domains using several plug-ins. The results obtained by IRIS are among the best reported in the literature. Our future work includes applying IRIS to more case studies from other domains and developing new plug-ins. Also exploring the relationship between IRIS and existing informal as well as formal approaches is part of our intended future work.

Reference

[1] J.O. Palmer, N.A. Fields, “An Integrated Environment for Requirements Engineering”, IEEE Software, pp. 80-85, 9(3), 1992.

[2] M. Shehata, A. Eberlein, "Requirements Interaction Management: A Multi-Level Framework", SEA 2002, The 6th IASTED International Conference on Software Engineering and Applications MIT, Cambridge, USA - November 4-6, 2002

[3] M. Shehata, A. Eberlein, "Issues in Requirements Reuse and Feature Interaction Management", ICSSEA 2002, Paris, France, December 3-5, 2002.

[4] M. Calder, E. Magill, M. Kolberg, S. Reiff-Marganic, “Feature Interaction: A Critical Review and Considered Forecast”, Computer Networks, Volume 41/1, pp 115-141, North-Holland

[5] R. Andrade, "Applying Use Case Maps and Formal Methods to the Development of Wireless Mobile ATM Networks". The Fifth NASA Formal Methods Workshop, Virginia, June 2000.

[6] Q. Fu, P. Harnois, L. Logrippo, J. Sincennes, "Feature Interaction Detection: a LOTOS-based Approach". Computer Networks 32, 4, (2000) 433-448.

[7] P. Zave, “Formal description of telecommunication services in Promela and Z”, In proceedings of the nineteenth International NATO summer school, 1999.

[8] M.Plath and M. D. Ryan, “The feature construct for SMV: semantics”, Sixth International Workshop on Feature Interactions in Telecommunications and Software Systems, IOS Press, 2000.

[9] B. Boehm, H., "Identifying Quality-Requirement Conflicts", IEEE International Conference of Requirements Engineering, IEEE CS Press, Colorado Springs, pp. 218, Apr. 1996.

[10] W. Robinson, S. Pawlowski, S. Volkov, “Requirements Interaction Management”, ACM Computing Surveys, Vol. 35, No. 2, June 2003, pp. 132–190.

[11] M. Shehata, A. Eberlein, "Requirements Interaction Detection using Semi-Formal Methods", ECBS 2003, 10th IEEE Symposium and Workshops on Engineering of Computer Based Systems, Alabama, USA April 2003.

[12] M. Shehata, A. Eberlein, "Detecting Requirements Interactions: A Three-Level Framework" , ASE 2003, Proceedings of the 18th IEEE Conference on Automated Software Engineering, October 6-10, 2003, Montreal, Canada

[13] M. Shehata, A. Eberlein, A. O. Fapojuwo, "Feature Interactions between Networked Smart Homes Appliances", QSSE 2004, 4th ASERC Workshop on Quantitative and Soft Computing Based Software Engineering, February 16-17 2004, Alberta, Canada

[14] R. Grosu, C. Klein, B. Rumpe, M. Broy, “State Transition Diagrams”, TUM-19630, 1996.

[15] K. J. Turner, “Formalizing the Chisel Feature Notation”, In: Sixth International Workshop on Feature Interactions in Telecommunications and Software Systems, Glasgow, Scotland, UK, May 2000. IOS Press, Amsterdam.

[16] D. Amyot, “Use Case Maps as a Feature Description Language”, In: S. Gilmore and M. Ryan (Eds), Language Constructs for Designing Features. Springer-Verlag. 27-44.

[17] M. Kolberg, E. H. Magill, D. Marples, S. Reiff, “Second Feature Interaction Contest”, The sixth international workshop on feature interactions in telecommunications and software systems (FIW’00), pp. 293-325, May 2000.

[18] M. Heisel, J. Souquière, “A heuristic algorithm to detect feature interactions in requirements” In S. Gilmore & M. Ryan, eds, ‘Language Constructs for Describing Features’, Springer-Verlag, 2000.

[19] M. Heisel, J. Souquière, “Detecting Feature Interactions - A Heuristic Approach”, In G. Saake and C. Türker, (eds), Proceedings 1st Fireworks workshop', Magdeburg, May 1998.

[20] M. Shehata, A. Eberlein, A. O. Fapojuwo, "The Use of Semi-Formal Methods for Detecting Requirements Interactions", SE 2004, February 17-19 2004, Innsbruck, Austria

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

[22] D. Briere, P. Hurley, “Smart homes for dummies”, 2nd edition, Wiley Publishing Inc, 2003, ISBN0764525395