UNIVERSITY OF CALGARY

Detecting Requirements Interactions Using Semi-Formal Methods

by

Mohamed Sami Abbass Shehata

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

CALGARY, ALBERTA

JULY, 2005

UNIVERSITY OF CALGARY

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled "Detecting Requirements Interactions Using Semi-Formal Methods" submitted by Mohamed Sami Abbass Shehata in partial fulfilment of the requirements of the degree of Doctor of Philosophy.

[illegible]

Supervisor, Dr. A. Eberlein,
Department of Electrical and Computer Engineering

[illegible]

Co- Supervisor, Dr. A. Fapojuwo,
Department of Electrical and Computer Engineering

[illegible]

Dr. R. Kremer,
Department of Computer Science

[illegible]

Dr. M. Husein,
Department of Chemical and Petroleum Engineering

External Reader, Dr. M. Reformat,
University of Alberta

22/7/05

Date

Abstract

Finding ways of detecting interactions between requirements is essential in order to develop a set of clear requirements, which serves as a foundation for successful software development. Detecting requirements interactions as early as possible helps avoid high repair costs.

This thesis presents IRIS, Identifying Requirements Interactions using Semi-formal methods, which is a semi-formal approach for detecting requirements interactions. IRIS is a systematic six step approach that uses tables, graphs, interaction detection scenarios, and subjective judgment to detect requirements interactions in software systems. IRIS has the advantage of not only being domain independent but also customizable towards a specific domain in order to enhance its performance. IRIS helps reduce the number of necessary pair-wise comparisons between requirements that have to be performed informally by a human expert. This reduction is achieved by discarding irrelevant comparisons that will not lead to interactions.

A general requirements interaction taxonomy was developed for identifying when two requirements are considered interacting. This requirements interaction taxonomy provides interaction detection scenarios that are used within IRIS for detecting interactions.

To validate IRIS, it was applied to three different case studies from different domains. In the first case study, the lift system, IRIS was able to detect 7 interactions as opposed to 6 interactions that were detected by another approach reported in literature. IRIS was also able to achieve 17.6% reduction in the number of comparisons. The second case study analyzed telephony features and IRIS was able to detect 21 interactions with 17.9% fewer feature comparisons. This result is very good as other approaches that detected 22 interactions all used formal methods. The third case study looked at smart homes policies. IRIS detected 83 interactions with 19.3% fewer policy comparisons. The smart homes case study is a major contribution as the results from it serve as the first fully documented analysis of interactions between smart homes policies in literature.

To facilitate the application of IRIS, a tool was implemented. IRIS-Tool Support (IRIS-TS) is built as an add-on module for DOORS which is a well-known commercial requirements management tool.

Acknowledgements

First of all, I want to thank God almighty for giving me the strength to pursue this degree. Looking back at the last four years, many people come to my mind who contributed in one way or another in this thesis.

I always considered that getting a Ph.D. degree is not only about conducting research, but rather it involves interacting with people and learning from their experience and mentalities. I was very lucky in working with two great supervisors, Dr. Armin Eberlein and Dr. Abraham Fapojuwo.

I would like to thank Dr. Armin Eberlein for his deep support and guidance in my research from inception through to completion. I'm really grateful to all the encouragement and inspiration that he gave to me.

I also want to thank Dr. Abraham Fapojuwo who helped me greatly during my research and provided me with huge valuable feedback during my research.

I also want to thank Dr. Rob Kremer for his feed back on this thesis.

Additionally, I would like to thank the Egyptian Higher Ministry of Education for the financial support that they covered me during my research course.

Moreover, I would like to thank all my fellow researchers in my research group, Abdallah Mohamed, Jiang Li, Simon Pfeiffer, Quin Zhang, and Majid Moussavi.

I would like also to thank Tim Yue who participated with me in creating the code for the tool support created in this thesis and presented in Chapter ~~10~~. 9

Finally, a big thank you goes to my family especially my parents and my wife Samar who continuously provided me with love, care, and encouragement during the period of this research.

Dedication

I would like to dedicate my thesis to:

My Parents: Sami and Nawal

My Wife: Samar

My Daughters: Dina and Noha

My Son: Ahmed

Table of Contents

List of Tables

List of Figures and Illustrations

List of Symbols and Abbreviations

3WC	Three Way Calling
AI	Artificial Intelligence
AOSD	Aspect Oriented Software Development
ATTR	Attribute
BCSM	Basic Call State Modes
CENELEC	European Committee for Electrotechnicl Standardization
CFBL	Call Forward on Busy Line
Ci	ith Constraint
COSPAN	COordination-SPecification ANalysis
CPL	Call Processing Language
CRESS	CHISEL Representation Employing Systematic Specification
CTL	Computational Tree Logic
CW	Call Waiting
DB	Data Base
DC	Duration Calculus
DDRA	Deficiency-Driven Requirements Analysis
DFC	Distributed Feature Composition
DOORS	A requirements management software tool by Telelogic Inc.
DXL	DOORS eXtension Language
EFSA	Extended Finite State Automata

EHS	European Home Systems
EIB	European Installation Bus
eSERL	extended Service Execution Rule Language
ESTI	European Standard Telecommunications Institute
ETS	Engineering Tool Software
FI	Feature Interaction
FIW	Feature Interaction Workshop
FIW00	The sixth Feature Interaction Workshop held in 2000
FOL	First Order Logic
FR	Functional Requirements
FSA	Finite State Automata
FSM	Finite State Machines
IN	Intelligent Networks
IRIS	Identifying Requirements Interactions using Semi-formal methods
IRIS-TS	IRIS – Tool Support
IVR	Interactive Voice Response
KAOS	Knowledge Acquisition in autOmated Specification of software
KNX	A smart homes networking system
LOTOS	Language Of Temporal Ordering Specification
LTL	Linear Temporal Logic

Lustre	A storage and file system architecture that comes from Linux and Clusters
MSC	Message Sequence Charts
NFR	Non-Functional Requirements
OO	Object Oriented
PIMM	Predefined Interaction Manager Module
POTS	Plain Old Telephone System
PROMELA	PROcess MEta LAnguage
PSHI DB	Predefined Smart Homes Interactions Data Base
PSTN	Public Switched Telephone Network
RACE	Research and Development in Advanced Communications in Europe
RBF	Ring Back when Free
RC	Reverse Charge
RE	Requirements Engineering
RIM	Requirements Interaction Management
ROSA	RACE Open Service Architecture
SCRi	ith interaction SCenaRio
SDL	Specification and Description Language
SE	Software Engineering
Si	ith interaction Subcategory
SMV	Symbolic Model Verifier
ti	ith interaction type

TL	Temporal Logic
TLA	Temporal Logic of Action
UCM	Use Case Maps
UML	Unified Modelling Language
VM	Voice Mail

CHAPTER ONE: INTRODUCTION

1.1 Introduction and Motivation of Research

Studies have claimed that in order to succeed in developing high-quality software systems, it is necessary to have correct and unambiguous requirements [1]. This makes requirements engineering (RE) a vital part of software development [2-5] and critical to the success of the entire project. Recent surveys by Nikula *et al.* [6] and McPhee [7] show that industry has started to realize the importance of good requirements engineering. Emam *et al.* [8] surveyed 56 projects worldwide over a period of two years and concluded that good requirements have a positive impact on the quality of software.

A key issue in obtaining a set of clear requirements is how to manage negative relationships between requirements [9] [10]. Robinson *et al.* [11] defines requirements interactions management as "the set of activities directed towards the discovery, management, and disposition of critical relationships among a set of requirements". Requirements often interact when developing new systems because of the heterogeneity and diversity of stakeholders [11] or because of reusing already existing requirements from previous similar projects where people make the assumption that the reused requirements will increase safety because they have been exercised extensively [12]. In either case, developing a software project should be done with an ongoing effort to discover and resolve interactions that could arise between requirements.

The so-called feature interaction problem has been extensively researched in the telecommunications domain to identify interactions between telephony features. A basic definition of feature interactions can be: Feature interaction is a situation where several features that are integrated on top of a base system may interfere and affect each other.

Features and requirements can be seen to have an n:m relationship. A high level requirement can consist of several features [13]. On the other hand, a feature may needs to be defined by several requirements. Hence, the relation between features and requirements can be seen as an n:m relationship.

Requirements interaction is similar to feature interactions in the sense that both try to identify the relationships between features or requirements. However, requirements interaction has a broader scope than the limited scope of feature interaction for the following reasons:

1. Requirements interaction considers non-functional requirements as well as functional requirements whereas feature interaction focuses on functional behaviour interactions.
2. The feature interaction research focuses primarily on the telecommunication domain where one examines possible interactions between new and existing telephony features. But requirements interaction is a phenomenon that can occur in any software domain.
3. Many of the current feature interaction approaches require design and possibly implementation-specific knowledge such as complete descriptions of all the states of the system. Such knowledge is not always available at the early requirements engineering phase. On the other hand, requirements interaction approaches focuses mainly on detecting interactions between requirements at the early requirements engineering phase.

4. While feature interaction tends to detect only technical behavioural interactions between requirements, requirements interactions additionally detects interactions between requirements caused by the heterogeneity of stakeholders.
5. The techniques used for the resolution of requirements interactions consider social and technical aspects. This means that the resolution of requirements interactions must involve stakeholders (e.g. the WinWin model [14, 15] involves stakeholders' views on the importance of each requirement and aims at achieving a win-win situation for all stakeholders involved). Whereas in feature interaction, most of the interaction resolution techniques assign priorities to the different features and the feature with the highest priority dominates.

In this thesis we focus on the broader area of requirements interactions. The motivation for this thesis was based on the following:

1. A review of the current practice of interaction detection (as summarized in Chapter 2) showed that there are two extremes: one extreme uses informal detection approaches using domain experts who rely on their experience with no systematic approach to follow. The other extreme uses formal approaches, such as the Specification and Description Language SDL [16]. However, domain experts are expensive, hard to find and prone to errors [15]. Formal approaches provide fairly accurate detection of interactions but not every company has the time and resources necessary to carry out a formal verification of their systems under development.
2. It appears as if there is currently no robust and complete definition of the different types of possible interactions between requirements. Most definitions describe, at

a very high level of abstraction, what interaction is without defining the different types of interactions or the various scenarios that can cause interactions. Furthermore, the different approaches surveyed in this thesis rely on detection of inconsistencies in formal models and design problems such as deadlocks and livelocks.

3. Many approaches and techniques have been proposed to solve the feature interaction problem in the telecommunications domain but only little effort has been spent on researching the applicability of possible solutions to this problem in other domains of software engineering.

1.2 Thesis Contributions

This Thesis offers 4 main contributions summarized as follows:

1.2.1 IRIS: A Semi-Formal Approach for Detecting Requirements Interactions

The first contribution of this thesis is the development of a semi-formal approach for detecting requirements interactions in software systems. This approach is termed IRIS which stands for Identifying Requirements Interactions using Semi-formal methods. IRIS has the following advantages over currently existing approaches:

1. Semi-formality of the approach: The proposed approach uses semi-formal methods for detecting interactions. This means that it uses tables, graphs, interaction detection scenarios, and subjective detection to detect interactions. This requires visual system representation and does not require any heavy mathematical modeling of the system under investigation as opposed to formal methods.

2. Detecting interactions at different levels of abstraction: The proposed IRIS approach uses attributes to represent system elements (e.g., dynamic behaviour requirements). This enables IRIS to detect interactions at different levels of abstraction. This thesis reports case studies in which IRIS was able to detect interactions at the requirements level (see Chapter 6), at the features level (see Chapter 7), and at the policies level (see Chapter 8).
3. Reduction in number of comparisons: IRIS reduces the number of necessary pair-wise comparisons that a human developer would have to perform between requirements in informal approaches and other semi-formal approaches described in the literature (more details on these approaches are provided in Chapter 2). IRIS achieves the reduction in the number of comparisons by discarding irrelevant comparisons that will not lead to interactions and focuses only on comparing requirements that are related either directly or sequentially. Hence, this can result in a clear reduction in the number of comparisons and consequently reduction in time and effort.
4. Domain independency: The proposed approach is not limited to a specific software domain (e.g., the well known telecommunications domain). To achieve this generality, IRIS was developed as a general approach that can be adapted to any software domain through a customization process to include specific knowledge about software domains through the use of plug-ins thus improving the detection success rate (see Chapter 5).
5. Extendability of the approach: IRIS was designed with a basic core and extension hooks. These extension hooks are insertion points that allow the addition of plug-

ins to IRIS basic core to extend its performance, increase the scope and thoroughness of interaction detection to include design and resource interactions, make IRIS applicable to new domains, and cope with any specific future needs by system developers.

1.2.2 Applying IRIS to detect Interactions in different domains

The second contribution of this thesis is the application of IRIS in detecting interactions in 3 case studies each belonging to different software domain. The first case study was done to detect interactions between the requirements of a lift system (control domain). The second case study was conducted to detect interactions between telephony features (telecommunications domain). The third case study was conducted to detect interactions between smart homes policies (policy domain). The first two case studies have been analyzed by other approaches and their results were reported in literature [17-19]. Hence, these results have been used as benchmarks to assess the effectiveness of IRIS. The third case study represents a major contribution in the interaction community as no complete interaction detection analysis between smart home policies has been reported in the literature.

1.2.3 A General Requirements Interaction Taxonomy

The third contribution of this thesis is the development of a general interaction taxonomy for classifying and identifying requirements interaction. The proposed taxonomy describes 9 main interaction categories, 24 interaction subcategories, 37 interaction types, and finally 37 interactions scenarios that also contain 37 interaction detection guidelines. The interaction detection guidelines help developers identify when two requirements are considered interacting. The proposed interaction taxonomy addresses the lack of detail that exists in other interaction taxonomies in the literature [20-22]. To validate the proposed interaction taxonomy, a comparison is made with other existing taxonomies in the literature. The results of the comparison show that the proposed interaction taxonomy was not only able to address the interaction issues in other taxonomies presented in the literature, but it also contained many other interaction types that have not been captured by other taxonomies (Chapter 3 provides more details).

1.2.4 A Tool Support for IRIS Integrated in the DOORS Requirements Management Software

To help software developers apply IRIS, a tool called IRIS-TS, which stands for IRIS Tool Support, was implemented as an add-on module for the commercial DOORS requirements management tool [23]. IRIS-TS appears as a drop down menu on the main tool bar of DOORS and has the ability to detect interactions between the requirements saved in DOORS. IRIS-TS performs a step-by-step walkthrough of the steps of IRIS and generates the appropriate inputs and outputs for the analyst. IRIS-TS was implemented using the DOORS eXtension Language (DXL).

1.3 Thesis Outline

This thesis contains ten chapters including the introduction chapter. The remaining nine chapters are organized as follows:

Chapter two contains surveys and analysis of the relevant literature. The literature review presents the necessary background information on the different approaches to interaction detection currently available.

Chapter three presents a general requirements interaction taxonomy used to identify when two requirements are considered interacting.

Chapter four presents the proposed semi-formal IRIS approach and describes in details its basic core.

Chapter five describes how IRIS can be customized and extended with plug-ins.

Chapter six presents the application of IRIS to the control domain. IRIS was used to detect interactions between the requirements of a lift system.

Chapter seven presents the application of IRIS to the telecommunications domain, i.e., the detection of interactions between telephony features.

Chapter eight describes the application of IRIS to the policies domain by analyzing smart homes policies for interactions.

Chapter nine presents the tool support, IRIS-TS, that was created to support the application of IRIS using the commercial DOORS requirements management software.

Chapter ten presents a summary of the thesis and the conclusions. Chapter ten also includes a list of future research topics on the work pioneered in this thesis.

At the end of the thesis, 6 appendices are included to provide complementary data and information to the thesis' main body.

CHAPTER TWO: CURRENT STATE OF THE ART

2.1 Introduction

Although there has been relatively little attention paid to the problem of detecting requirements interaction in software systems, the feature interaction problem has been very well researched in the telecommunications domain. To provide the necessary literature review that the work in this thesis was based on, some of the more relevant previous work is described in this chapter. It is worth mentioning that other relevant previous work on requirements interaction taxonomy and on the case studies presented later-on in this thesis are specified in the appropriate chapters. Hence, this chapter focuses only on the previously developed approaches for interaction detection.

The structure of this chapter is as follows: Section 2.2 presents a survey on the current state of the art regarding approaches in the feature interaction research community. Section 2.3 gives a survey on the current state of the art regarding approaches from the requirements engineering research community. Finally, section 2.4 provides a summary of this chapter.

2.2 Surveying the Feature Interactions Area

2.2.1 The Feature Interactions Problem

Feature interaction is a situation where several features that are integrated on top of a base system may interfere with each other, or interact in ways that are hard to predict. To explain the feature interaction problem, consider the following two telephony features: Call Waiting (CW) and Call Forward on Busy Line (CFBL) [19, 24]. The CW feature is a feature when active allows the subscriber to be notified of an incoming call while s/he is busy and to accept the new call by putting the original call on hold. Then s/he is able to toggle between the two calls. The CFBL feature, when active, will redirect all incoming calls to the subscriber phone number to a predefined number when the subscriber line is busy. The interaction occurs when these two features are implemented and activated on the same phone line. In this case the system is unable to decide what to do: should it notify the user of the incoming call and allow him to accept it according to the CW feature, or it should automatically forward the incoming call to the predefined phone number according to the CFBL feature.

The features interactions problem has received a lot of attention from the telecommunications industry where many approaches have been developed. A good description of the current research status of feature interaction in telecommunications and software systems can be found in the proceedings of the feature interaction workshops [25-31]. However, in this section, we try to summarize some of the work done on creating approaches for detecting features interactions.

2.2.2 Methodology for Surveying the Feature Interaction Approaches

In this section, different approaches for detecting feature interactions are presented. A classification can be made for the different feature interaction approaches based on whether an approach is a static offline detection approach or it is a run-time detection and resolution approach (also called online approaches). In this section we only focus on offline approaches which are more relevant and of interest to the proposed IRIS approach than the online approaches.

A classification of the offline approaches can be made based on their formality. According to the formality criteria, offline detection approaches can be classified based on their degree of formality into the following two categories: Semi-formal and formal approaches. Semi-formal approaches use tables, graphs, and human subjective detection (7 approaches) while formal approaches use formal methods (59 approaches). It must be noted that an approach can have more than one paper published on it, however, all these papers are counted only once as they all relate to the same approach. Due to the relevance of semi-formal approaches to this thesis, all surveyed semi-formal approaches in the literature are described in detail.

In the formal approaches category, only a summary table of the approaches is first presented, then some of the approaches are described in detail. The selection criteria for describing a formal approach in detail will depend on whether the approach has been successfully applied in the industry or if the approach has a major impact in the feature interactions research community.

The survey presented in this section is based on the following resources:

1. An extensive survey conducted by the author using online resources on the World Wide Web and online database libraries such as the IEEE [32], ACM [33], and CITESEER [34] digital libraries.
2. The proceedings of the feature interactions workshops [25-31] and journals special issues on feature interactions [35].
3. The survey by M. Calder *et al.* [36] regarding the different approaches in the area of feature interactions.
4. The survey by Keck and Kuehn [37] on the feature interaction problem in telecommunications systems.

2.2.3 Detecting Features Interactions using Semi-Formal Approaches

Semi-formal approaches create and use graphical and tabular notations for representing the system and using these representations for detecting interactions without the need to use formal models. Through the conducted survey, only seven approaches were found to fall under this category. In the following, all of the seven semi-formal approaches are presented. Each approach will be described in a table using the following items: the heading of the table is used to give an ID for the approach and also to list the authors and references of the approach, the notation used in the approach, the main idea of the approach, steps of the approach to describe how the approach is executed, results to describe if the approach has reported any case studies or industrial results, types of interactions that can be detected by this approach, pros to describe the points in favour of this approach, and finally criticisms to list the points against and limitations of this approach.

It is worth mentioning that all of these approaches and IRIS are similar in using semi-formal methods. However, there are a number of differences between IRIS and these approaches (see the criticism row in the tables below). Also, a summary of the differences is listed in Chapter 4.

Table 2.1: The Approach by Wakahara, Fujioka, Kikuta, Yagi, and Sakai

SF 1: The Approach by Wakahara, Fujioka, Kikuta, Yagi, and Sakai [38]	
Notation Used	Message Sequence Charts (MSC)
Approach Main Idea	The main idea of the approach is to analyze the input-output relationships between the features of the telecommunications domain. The analysis is done using human experts analyzing message sequence charts
Steps of the Approach	• Informally specify features • Check for features completion using specific telecomm. knowledge about how a feature should be written • Define obvious interactions between features due to explicit input-output relationships between the features • Define implicit relationships between the features using impact knowledge of features in telecomm. domain • Develop MSC for system and features by adding all MSC of features and system in one chart • Detect interactions by having an expert inspecting the MSC with the help of a telecommunications features knowledge database
Results	Examples from the telecommunications telephony features
Types of Interactions Detected	The interactions detected are in the form of: • Duplication • Redundancy • Incorrect order of execution • Inconsistency • Vagueness/non-determinism • Looping
Pros	• Simple to use • Do not require complete specification details to be applied but rather missing details are completed during the execution of the approach • One of the early attempts to tackle the problem of feature interactions using semi-formal methods
Criticism	• Specific to the telecommunications domain due to the nature of knowledge being used • The database and knowledge used are very abstract • Combing two or more features on top of the base system in one MSC chart is not easy as the resulting MSC will be hard to analyze by an expert • Detects only interactions due to input-output relationships problems whereas many other types of interactions are ignored • Do not address system properties that must be preserved (e.g. non-functional aspects such as availability) • Do not address resource related interactions

Table 2.2: Approach by Mierop, Tax, and Janmaat

SF2: The Approach by Mierop, Tax, and Janmaat [39]	
Notation Used	Object Oriented (OO)
Approach Main Idea	The main idea of the approach is to represent the system and the features as objects with interfaces. During the specification of features as objects, ambiguities that arise are considered as interactions between the features
Steps of the Approach	• Build an object oriented environment for the telecommunication system and represent users in this base environment as objects. Each user object will have an interface, a user agent object, and a user profile object • Specify the features to be added to the base system and model them as scenarios on the object oriented environment • Human developers analyze the object oriented environment and the object oriented feature specification for any ambiguous situations such as two services inducing a non-determinism on busy signal
Results	CW and CFBL example from European Community research Project RACE Open Service Architecture (ROSA)
Types of Interactions Detected	The interactions detected are in the form of ambiguity in specification of the features in the object oriented model
Pros	• Separation of feature interactions from other resource interactions • New and original representation of the problem • One of the early attempts to tackle the problem of feature interactions using semi-formal methods
Criticism	• No proof of application outside the telecommunications domain • The representation of the telecommunications domain in object oriented notation is not an easy task and the approach is therefore did not spread • Detects only interactions due to ambiguous situations which was defined as non-determinism transitions due to invoking more than one feature by a common signal • Do not address system properties that must be preserved (e.g. non-functional aspects such as availability) • Do not address resource related interactions

Table 2.3: The Approach by Kimbler, Kuisch, and Muller

SF 3: The Approach by Kimbler, Kuisch, and Muller [40]	
Notation Used	None
Approach Main Idea	The features are categorized into categories based on the similarities of their nature (e.g., charging features) and the similarities of the roles they play. Interaction-prone feature combinations are obtained when two categories are said to be interaction-prone which is decided based on the roles and resources that the categories play and use. Once non interaction-prone combinations are eliminated, the rest are analyzed using a systematic approach for identifying interactions. The interaction detection is based on manually analyzing the features service life cycle created by the European Standards Telecommunications Institute/Group 6 (ESTI/NA6). The analysis is based on executing four steps in sequence and manually detecting interactions between features.
Steps of the Approach	• Analyse interactions between service pairs • Analyse combinations of feature categories • Once irrelevant non-interaction prone combinations are discarded, compare remaining stand alone feature pairs • Compare feature pairs within service context by manually analyzing the feature specifications provided by the ESTI/NA6 for interactions
Results	No
Types of Interactions Detected	Negative impact of a transition of the first feature on any state of the second feature
Pros	• Uses experience along with structure approach for detecting interactions • Analyze features in the context of their services in addition to the stand alone analysis
Criticism	• Especially designed for the telecommunications domain • The final interaction detection relies totally on experience with no rules or guidelines • The approach used some serious simplifications in the ESTI/NA6 specification with no proof of validity (e.g., the modification of invocation data state cannot cause any interactions)

Table 2.4: The Approach by Dankel, Schmalz, Walker, Nielsen, Muzzi, and Rhodes

SF 4: The Approach by Dankel, Schmalz, Walker, Nielsen, Muzzi, and Rhodes [41]	
Notation Used	High level predicated
Approach Main Idea	The main idea of the approach is to use a feature capturing system which will accept natural language statements from the designer regarding specification of new features and then convert it to high level predicates. The developed predicated are added or used to update a knowledge base. Artificial intelligence is used to announce any ambiguities in the new specifications. Finally, developed models are shown to designers to decide if there are interactions between the newly added feature and other existing features
Steps of the Approach	• The designer input natural language statements about the new feature through a graphical interface • Parse the statements using a lexical and grammar knowledge parser • Generate high level predicates for the parsed statements • Pass predicates to command interpreter to find any ambiguities to be returned to the designer. if no ambiguities are found, add or update the knowledge base with the new feature • Generate graphical models of the system for the new features with other features based on the selection of the designer to check for interactions • Human designer checks the models for interactions
Results	No
Types of Interactions Detected	Incompatibility between two features
Pros	• Allows designers to specify features with natural language • Uses artificial intelligence to remove easy to detect ambiguities
Criticism	• No proof of applicability outside the telecommunications domain • The actual detection of interactions is completely done by human and with experience • The approach does not have any systematic steps in it and does not address the detection of interaction

Table 2.5: The Approach by Kuisch, Janmaat, Mulder, and Keesmaat

SF 5: The Approach by Kuisch, Janmaat, Mulder, and Keesmaat [42]	
Notation Used	Basic Call State Modes (BCSM)
Approach Main Idea	The main idea of the proposed approach is to represent the system and the features using a template. This template contains information about the functionalities of features through the representation in BCSM notation and the use of Detection points, information flows, and resources. The human developers analyze the BCSM for interactions using specific criteria
Steps of the Approach	• Produce a behavioural specification according to a predefined template • Specify the features to be added to the base system using the BCSM, the Detection points, the dataflow, and the resources usage • Combine features to be examined for interaction in one model • Determine the range that each feature controls on the BCSM model • Allow human detection of interaction using the criteria that interaction occurs when there is a conflicting overlap between the ranges of two features or when the two features want to process each others flow data in conflicting manner
Results	Examples from the telecommunications domain
Types of Interactions Detected	The interactions detected are in the form of: • Conflicting data manipulation • Conflict of control due to overlapping of features range • Shared resources interactions
Pros	• Practical and have sufficient in-depth details about the telecommunications IN networks • Considers resources interactions • One of the early attempts to tackle the problem of feature interactions using semi-formal methods
Criticism	• Specific to the telecommunications domain • Specification in BCSM is not an easy task • The reference does not describe types of resource related interactions that can be detected but states that it is limited and needs further development • Do not address system properties that must be preserved (e.g. non-functional aspects such as availability)

Table 2.6: The Approach by Keck

SF 6: The Approach by Keck [43]	
Notation Used	Basic Call State Modes (BCSM)
Approach Main Idea	Detect interactions prone scenarios by a tool that generates a list of interaction prone scenarios based on a criteria for interaction detection. The generated list can be further analyzed by other interaction approaches
Technique of the Approach	The developed tool will detect interaction prone scenarios using the following components: • Initialization Components: This component select and parse the provided service description (i.e., provides pairs of services to be examined) • Filtering Component: This component applies different filters based on criteria used for identifying scenario prone interactions • Result Generation Component: This component creates a file reporting the results of applying the filters
Results	Case study on the telecommunications telephony features
Types of Interactions	Filtering: Trigger collision interactions, Resource conflict interactions, and Data conflict interactions
Pros	Reduces the number of scenarios to be examined in large and complex systems where analysis of all behavioural scenarios of the system is hard
Criticism	• The tool requires design details on the behaviour of features to be examined. The features are then written using the BCSM notation • The generated list contains only interaction prone scenarios and this list must be analyzed by another detection approach for deciding which features are really interacting • The criteria used for identifying interaction prone scenarios are limited and many interactions (e.g. sequential interactions) are not addressed • Do not address system properties that must be preserved (e.g. non-functional aspects such as availability)

Table 2.7: Approach by Kimbler and Sobirk

SF 7: The Approach by Kimbler and Sobirk [44]	
Notation Used	Use Case Models
Approach Main Idea	The main idea of the approach is to build a use case model that describe the different scenarios of using the system and then build a service usage model that describes the dynamic relations of the features from the users point of view. A human expert have then to manually analyze the models
Steps of the Approach	• Create a use case model that have many use cases that describe roles and actors for the system and how different scenarios of the system might be executed by the actors • Transform the use case model into a service usage model that describes the dynamic behaviour of the service form the user's perspective • In the service usage model, create service usage graphs that is based on state diagrams • The process of building the service usage model is done as follows: first the use cases from the use case model are analyzed, then the informal description of each use case is converted into sequence of events, then identify system usage states, and finally combine these analyzed data into a service usage model • Manually analyze the created models for interaction-prone features by a human expert • Two features are considered interaction-prone when they access or modify same service or call specific data
Results	Examples from the telecomm domain
Types of Interactions Detected	Incompatibility between two features due to shared service or data access violation
Pros	• Uses experience along with structure approach for detecting interactions • Analyze features in the context of their services in addition to the stand alone analysis • The approach can avoid state explosion by dividing the use case graphs into smaller ones
Criticism	• The created use cases, which is the first step and basic core, cannot cover all possible usage scenarios • The final interaction detection relies on experience with limited definition of when two features interact • Creating use cases for new systems is very hard and hence the authors explicitly limit the approach to telecommunications domain • The criteria used for identifying interaction prone scenarios are limited and many interactions (e.g. sequential interactions) are not addressed • Do not address system properties that must be preserved (e.g. non-functional aspects such as availability)

2.2.4 Detecting Feature Interactions using Formal Approaches

A formal approach can be simply defined as an approach that uses a formal language for describing software specifications such that formal proofs are possible about the software specification. A formal language is a language whose vocabulary, syntax, and semantics is based on mathematical concepts whose properties have been well investigated and are well understood [45].

In this section we present a summary of the formal approaches for detecting feature interactions and highlight some of the famous approaches that have been either applied in the industry or have significant impact in the feature interaction research community.

Generally, formal approaches for detecting features interactions can be divided into the following two sub-categories:

1. Approaches that employ Specific software engineering techniques: These approaches employ techniques inspired by the software engineering domain and use different formal languages (e.g., SDL [16, 46] and LOTOS [47]),
2. Approaches that employ formal methods: These are approaches that mainly use logic and formal languages like SDL to validate properties and/or behaviour. These approaches are in turn divided into:
    a. Properties only approaches
    b. Behavioural only approaches
    c. Properties and behavioural approaches

Software engineering approaches are often considered part of formal approaches since they always involve a formal language in association with the software engineering approach they adopt. However, a difference between software engineering approaches and approaches that employ formal methods is that the latter approach uses only logic and formal languages, whereas software engineering approaches use a more comprehensive software engineering view.

2.2.4.1 Approaches Employing Specific Software Engineering Techniques

Software engineering approaches use specific software engineering techniques that have been used elsewhere in the area of software engineering. Usually, software engineering approaches use a formal language to detect and eliminate interactions between features. Table 2.8 lists some of the approaches that have been introduced in the feature interaction community that fall in this category.

Table 2.8: Software Engineering Approaches

ID	Approach Authors and References	Software Technique used	Formal Notation used	Application Phase	Reported Results
F1	Hay, Atlee [48]	Feature Composition	Labeled Transition Diagrams	Design	No
F2	Braithwaite, Atlee [49]	Layered State Transition	State Machines	Specification	Case study (telecomm. domain)
F3	Kelly, Crowther, King, Masson, DeLapeyre [50]	SDL	SDL	Specification	Case study (telecomm. domain)
F4	Bredereke [51]	Product Families	CSP-OZ	Requirements	Case study (telecomm. domain)
F5	Heisel, Souquieres [17, 18]	Requirements Elicitation	System state traces	Requirements	Case study (lift system)
F6	Zave, Jackson [52-55]	Feature Architecture	DFC	Design and implementation	Industrial scale (telecomm. domain)
F7	Iraqi, Erradi [56]	Composition of FSA	MONDEL	Specification	Case study (telecomm. domain)
F8	Prehofer [57]	Feature Oriented Programming	JAVA	Requirements	Examples (telecomm. domain)
F9	Utas [58]	Pattern Languages	FSM	Implementation	Industrial Scale (Telecomm. domain)

Table 2.8 - Continued: Software Engineering Approaches

ID	Approach Authors and References	Software Technique used	Formal Notation used	Application Phase	Reported Results
F10	Blair, PANG [59]	Aspect Oriented Software Development	Aspect J	Specification	Case study (email system)`
F11	Amyot, Charfi, Corse, Gray, Logrippo, Sincennes, Stepie, Ware [60]	Use Case Maps	LOTOS	Requirements, Design	Industrial scale (telecomm. domain)
F12	Prehofer [61]	Feature Composition	State chart diagrams	Design	Examples (email system)
F13	Berkani, Cave, Coudert, Kaly, Le Gall, Ouabdesselam, Richier [62]	Service Integration	State Transition Rules	Design	Examples (telecomm. Domain)
F14	Metzger, Webel [63, 64]	Traceability relationships	Formal product model (uses SDL)	Requirements, Strategy, Structure, Environment	Case study (heating and illumination control system)
F15	Turner [65-68]	CHISEL	LOTOS, SDL	Requirements	Examples on Interactive Voice Services (IVS)
F16	Zave [69, 70]	Component architecture	DFC	Design	Examples (telecomm. Domain and Email system)
F17	Choi, Kim, Lee, Kwon [71]	Distributed functional plan abstraction level	Petri nets	Design	Case study (telecomm. domain)
F18	Bredereke [72, 73]	Automata theoretic formalization	ESTELLE	Design	Case study (telecomm. domain)
F19	Klein, Prehofer, Rumpe [74]	Feature Composition	State Transition Diagrams	Design	Example (telecomm. domain)
F20	Faci, Logrippo [75]	Goal oriented knowledge	LOTOS	Design	Examples (telecomm. Domain)

In the following, more detailed explanation of the approaches F6, F9, and F11 is presented (industrial scale application). Also, detailed explanation of the approaches F14 and F15 is presented (impact on feature interactions research community)

Table 2.9: The Approach by Zave and Jackson

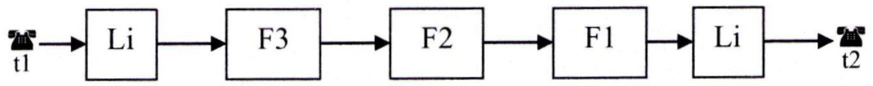
F 6: The Approach by Zave and Jackson [52-55]	
Notation Used	Distributed Feature Composition (DFC)
Approach Main Idea	Use the known pipe and filter principle. In this principle: • Filters communicate with the environment through only pipes • Filters does not know what is on the other side of the pipe Based on this principle, feature interactions can be prevented through the enforcement of a certain architecture called the DFC on the telecommunications network
Technique of the Approach	A structure is used on which a line interface is used to represent interface between a telephone and the network through certain ports on each interface. Features are represented by feature boxes. On call request to t1, the interface of t1 sets up an outgoing call which goes to F3. Now, F3 generates an outgoing call to F2 and wait for response. Also, F2 generates an outgoing call to F1 and wait for response. The last feature who receives the back signal is F1 and therefore if it is a trigger to F1then F1 will make changes according to its specifications. But if F1 is not triggered or it is disabled, then F2 is to be checked if it is triggered and so on. Hence it is clear that F1 has the highest priority and then F and finally F3 
Results	Industrial application in the telecommunications domain (AT&T Inc.)
Types of Interactions Detected	Prevention of interactions through the enforcement of the DFC architecture

Table 2.10: The Approach by Utas

F 9: The Approach by Utas [58]	
Notation Used	Finite State Machines (FSM)
Approach Main Idea	The main idea of the paper is to present a pattern language for tackling the problem of feature interactions. A pattern language is a collection of patterns that are used to solve a set of related problems (in this case feature interactions). A pattern is a general technique used to tackle a problem using a standard form. In this approach, each pattern can be used to handle a group of similar interactions (e.g., the pattern called PFE Chain of responsibility is used to tackle interactions that arise when two features can trigger at the same feature alternation point which is the time when a feature modifies the base system).
Technique of the Approach	The approach technique is based on developing several patterns to handle the different feature interactions at the implementation level. These patterns are then applied to detect and resolve feature interactions. Each pattern will consist of: Context, Problem, Forces, Solution, Rationale, Resulting Context, Examples, and Related Patterns.
Results	Industrial application in the telecommunications domain (Nortel GSM Mobile switching centre)
Types of Interactions Detected	• A feature that changes a basic call parameter that must be used by another feature • Interactions between a feature monitoring a channel data and features that can modify the users connection (e.g., close the channel) • Interactions between a feature that needs to change the state of another feature to execute • A feature that needs to perform a query and wait for response and depending on the result of the query suitable actions can be done • Interactions between multiplexer features that can run at the same time for the same user • Interactions between features that are triggered at the same feature alternation time • Interactions between an feature active feature and another incompatible feature that is being triggered

Table 2.11: The Approach by Amyot *et al.*

F11: The Approach by Amyot, Charfi, Corse, Gray, Logrippo, Sincennes, Stepie, and Ware [60]	
Notation Used	Use Case Maps (UCM), LOTOS
Approach Main Idea	UCM provides very good visual representation for features and can be used as a front end for any formal language. In this work, LOTOS was chosen as the formal language that is used to do the interaction detection. The approach first uses UCM to visually represent features then the generated representations are translated into LOTOS to be validated for interactions
Steps of the Approach	• Represent the system features using UCM • Translate the generated UCM models into LOTOS manually then automatically • Extract test scenarios putting in mind that the tests should test: basic system properties, individual features properties, and interactions between features • Use acceptance/rejection test scenarios to execute the LOTOS specification and see if the LOTOS specification accept or reject the test (reject means the executed behaviour of the specification does not match the expected behaviour specified in the original test scenario)
Results	Industrial results in the telecommunications domain (Mitel Inc.)
Types of Interactions Detected	UCM itself does not provide results of interactions unless translated into a formal language. However, once translated into LOTOS, the interactions found would be in the form of: • Basic service properties violation • Scenarios that show negative impact of one feature on another feature

Table 2.12: The Approach by Turner

F 14: The Approach by Turner [65-68]	
Notation Used	Chisel Representation Employing Systematic Specification (CRESS), LOTOS, and SDL
Approach Main Idea	The CRESS notation can be used to graphically represent a service and its features. Then developed graphical notation can be translated into a formal language (either SDL or LOTOS) where interaction detection takes place
Technique of the Approach	• Apply CRESS to represent the service and its features • Translate the generated graphical representation into SDL or LOTOS using SDL or LOTOS code generators • Analyze the features using either an SDL validator or a LOTOS validator to find problems either in the features themselves or in the group behaviour of the features
Results	Examples are given on detecting interactions in Interactive Voice Services (IVS)
Types of Detected Interactions	CRESS itself does not provide results of interactions unless translated into a formal language. However, once translated into SDL or LOTOS, the interactions found would be in the form of inconsistency and deadlocks

Table 2.13: The Approach by Metzger and Webel

F 15: The Approach by Metzger and Webel [63, 64]	
Notation Used	Formal product model (uses SDL)
Approach Main Idea	The approach main idea is to detect interactions caused by the environment as well as interactions caused by the system. The approach is based on developing a formal product model that describes requirements, functional needs, tasks, and functional strategies. Using this formal model, analysis can be made to detect interactions based on dependencies between the functional needs and the other artifacts of the formal product model.
Technique of the Approach	• Develop the formal product model of the system • Detect interactions at the requirements level by developing a dependency graph between the needs and the tasks. An interaction point is a node that realizes more than one need and has more than one direct parent. From these points of interaction, the actual interactions can be deduced. • To refine the potential interactions detected at the requirements level, interactions that cannot occur should be eliminated. This is done by only considering tasks that are directly and not transitively realized by the task at the point of interaction. • Detect interactions at the strategy level. After the above levels of information have been considered, dependencies between tasks that are introduced by their realization can be examined as soon as the developers have specified the strategies of the respective tasks. Dependencies on this level can occur because strategies can be coupled by signals or attributes to exchange information • Finally detect interactions at the environmental level. This is done by considering the dependencies that arise due to the environment (e.g. building architecture)
Results	Case studies in the building control system, automotive control system, and railway crossing controller
Types of Detected Interactions	Negative dependencies

2.2.4.2 Approaches Employing Formal Methods

Approaches that employ formal methods are divided into three categories:

- Property only approaches: This category contains approaches that represent the features and the base system in terms of abstract properties and then check for interactions such as inconsistencies or unsatisfiabilities.
- Behavioural only approaches: This category contains approaches that describe features and the base system in terms of behavioural models and then check for interactions such as non-determinism and deadlocks.
- Properties and Behavioural approaches: The third category contains approaches that describe features and the base system in terms of both properties and behavioural models and then check for interactions such as: combined features do not satisfy the corresponding combined properties (i.e., a property of a feature can be satisfied in the behavioural model of the feature but when two combined features are modeled together in one behavioural model, the combined properties of the two features are not satisfied)

2.2.4.2.1 Properties Only Approaches

Table 2.14 presents a summary of some of the approaches that use properties to detect interactions. Detailed explanation of the approaches F21, F22, and F23 are presented later on after Table 2.14.

Table 2.14: Properties only approaches

ID	Approach authors	Property language	Detected Interactions
F21	Blom, Jonsson, Kempe [76]	TLA	Deadlocks and Inconsistencies
F22	Gibson [77, 78]	FOL & TLA	Invariant violations
F23	Felty, Namjoshi [79]	LTL	Inconsistencies
F24	Rochefort, Hoover [80]	Constructive Logic	Satisfiability
F25	Frappier, Mili, Desharnais [81]	FOL	Inconsistencies
F26	Bostrom, Engstedt [82]	DELPHI	Inconsistencies
F27	Calder, Miller [83]	LTL	Deadlocks, race conditions
F28	Lee [84, 85]	Object-Z	State variables conflicts
F29	M. Butler [86]	Z	Inconsistencies

Table 2.15: The Approach by Blom, Jonsson, and Kempe

F 21: The Approach by Blom, Jonsson, and Kempe [76]	
Notation Used	Temporal Logic of Action (TLA)
Approach Main Idea	A service is considered as a module which can be formalized. Also each feature is seen as an independent formal module. The overall system is obtained by composing the service and features modules. This composition is seen as the conjunction of the properties of the modules.
Technique of the Approach	• Specify the basic system using TLA in the form of variables, events, restriction, initial condition, and reaction part, • Check for deadlocks interactions in the base system by making sure that the system always reaches states where other events can still occur • Specify features using TLA • Check for logical inconsistencies of (Feature_A AND Feature_B) over the base system • Resolve interactions between the interacting features by having Feature_A or Feature_B weaker
Results	Case study in the telecommunication domain on telephony features
Types of Interactions Detected	• Deadlocks • Logical inconsistencies between actions exhibited by two features

Table 2.16: The Approach by Felty and Namjoshi

F 22: The Approach by Felty and Namjoshi [79]	
Notation Used	Linear Temporal Logic (LTL)
Approach Main Idea	Base system and features are specified in LTL. Two features are considered interacting if their specifications ar mutually inconsistent under axiom properties about the underlying base system behaviour
Technique of the Approach	• Model the base system axiom properties using LTL and W-automata • Model the features specifications using LTL and W-automata • Use the model checker COSPAN to check for consistency of the modeled formulae • Two features A and B interact IFF A and B can be enabled together under the system properties such that: (System prosperities hold) AND (A and B are enabled together) AND (Some feature property doesn't hold)
Results	Case study in the telecommunication domain on 10 telephony features based on Bell-labs specifications documents
Types of Interactions Detected	• Inconsistencies between logical formulae

Table 2.17: The Approach by Gibson

F 23: The Approach by Gibson [77, 78]	
Notation Used	Temporal Logic of Action (TLA)
Approach Main Idea	The main idea of the approach is that the base system and the features can be treated as objects. The author used TLA to express the liveness properties (such as always, eventually). The liveness properties are then checked for each pair of features to detect interactions.
Technique of the Approach	• Model the base system axiom properties using fair objects semantics (which is TLA and Object Oriented concepts) • Specify state invariant properties (properties that contain the word "Always") and fairness properties (properties that contain the word "Eventually") • Classify the features under consideration according to their triggers using a triggered feature taxonomy • According to the result of the classification, apply interaction detection technique to mainly detect no-determinism • Resolve interaction based on prioritizing the features
Results	Examples from the telecommunication domain on telephony features
Types of Interactions Detected	• Invariant violations

2.2.4.2.2 Behavioural Only Approaches

Table 2.18 presents a summary of some of the approaches that use behavioural languages to represent the base system and the features to check for interactions. The different approaches detect different interactions and at different levels of abstraction. A detailed explanation is given after Table 2.18 on the approaches F30, F32, and F37.

Table 2.18: Behavioural only approaches

ID	Approach authors	Behavioural language	Detected interactions
F30	Hall [87]	State Transition Diagrams	Inconsistent state changes, Inconsistent actions
F31	Plath, Ryan [88]	CSP	Deadlocks
F32	Bruns, Mataga, Sutherland [89]	Chisel variant	Order dependency
F33	Blom [90]	MSC variant	Inconsistent post-conditions Inconsistent event
F34	Au, Atlee [91]	State Transition Machines	Control and data modification. Resource contention, Unreachable states
F35	Bergstra, Bouma [92]	Synchronous MSC	Inconsistencies
F36	Laporta, Lee, Lin, Yannakakis [93]	FSA	Language differences
F37	Khoumsi, Bevelo [94, 95]	ESFA	Non-determinism, Inconsistencies
F38	Inoue, Takami, Ohta [96, 97], Harada, Hirakawa, Takenaka [98, 99]	State Transition Rules	Abnormal state, Transition disappearance of normal state
F39	Nakamura, Kakuda, Kikuno [100]	FSM	Deadlocks, Loops, Non-determinism
F40	Thistle, Malhame, Hoang [101]	Control theory	Conflicting languages

Table 2.18 - Continued: Behavioural only approaches

ID	Approach authors	Behavioural language	Detected interactions
F41	Chan, Bochmann [102]	SDL, MSC	Resource contention, Incoherence, non-determinism, deadlocks, livelocks
F42	Mitchell, Thomson, Jervis [103]	MSC, Process Algebra	Phase transition interactions
F43	Kawauchi, Ohta [104]	State Transition Rules	Three way interactions
F44	De Marco, Khendek [105]	eSERL	Inconsistency of composition
F45	Aggoun, Combes [106]	SDL	State errors
F46	Lin, Lin [107]	PROMELA	Violation of assertions
F47	Nakamura, Leelaprute, Matsumoto, Kikuno [108, 109]	CPL	Semantic warnings

Table 2.19: The Approach by Hall

F 30: The Approach by Hall [87]	
Notation Used	State Transition Diagrams
Approach Main Idea	The main idea of the approach is to use foreground/background models for basic system and features combination to detect interactions. A background model is a model used to represent the base system and is of low priority whereas a foreground model is a model of the feature that has higher priority and when merged with a background model will override only specific parts of it and inherit its un-ridden parts. The combination approach is based on building a model for the basic system extended for feature F1 and another model for the basic system extended for F2 and then use a straight merge to combine the two models. An interaction would occur whenever the conceptual foreground behaviour of a feature is inconsistent of the conceptual background or default behaviour of another feature. The resolution technique is to allow the foreground models to override the behaviour of the background models at points of interaction. However, un-overridden points are left as is
Technique of the Approach	• Construct a foreground model of each feature • Construct a background model of the base system • Validate ($FG\|_{F1}$ AND BG) • Validate ($FG\|_{F2}$ AND BG) • Merge only the foreground models of the features using direct merge to get a new foreground model of $F1 \oplus F2$ • Validate ($FG\|_{F1 \oplus F2}$ AND BG) • Resolve any interactions detected from the previous step as explained earlier
Results	Case study in the telecommunications domain based on the second feature interaction contest [19]
Types of Interactions Detected	• Type I interactions occur when feature combination results in an ill-defined next state or output function for the resulting reactive system • Type II interactions occur when feature combination results in the violation of a correctness property for one of the individual features

Table 2.20: The Approach by Bruns and Mataga, Sutherland

F 32: The Approach by Bruns and Mataga, Sutherland [89]	
Notation Used	CHISEL variant
Approach Main Idea	The main idea of the approach is based on having an original service and then applying a new feature to the service to extend the base service. Features are implemented on top of the base system by adding them in an ordered sequence. If the system behaves differently when the order of the features are changed then these features are considered as interacting
Technique of the Approach	• Model the base system as a state transition system • Model the features that consists of sequence of updates, sequence of reactions, and sequence of events • Apply F1 then F2 to the system and capture the behaviour of the system • Reverse the order and apply F2 then F1 and capture the behaviour of the system • Interaction occurs if the system behaviour is different, i.e., $(F1(F2(S))) \neq (F2(F1(S)))$
Results	Case study in the telecommunications domain based on telephony features
Types of Interactions Detected	Order sensitive

Table 2.21: The Approach by Khoumsi and Bevelo

F 37: The Approach by Khoumsi and Bevelo [94, 95]	
Notation Used	Extended Finite State Automata (EFSA)
Approach Main Idea	The main idea of the approach is inspired from the control theory of discrete events. Features can be described using EFSA while the system can be extended using the finite state automata (FSA). Features are added on top of the base system to extend it. The extended system is then checked for non-determinism or variable inconsistencies
Technique of the Approach	• Model the base system using FSA • Model the features using the EFSA • Describe a scenario that express the non occurrence of the suspected interaction • Transform EFSA to FSA • Apply a model checker to see if the scenario holds
Results	Case study in the telecommunications domain based on the second feature interaction contest telephony features [19]
Types of Interactions Detected	• Non-determinism • Inconsistencies

2.2.4.2.3 Properties and Behavioural Approaches

Table 2.22 presents a summary of some of the approaches that use both property and behavioural languages to represent the base system and the features to check for interactions. A detailed explanation is given after Table 2.22 on the approaches F50, F51, and F55.

Table 2.22: Properties and behavioural approaches

ID	Approach authors	Property language	Behavioural language
F48	Combes, Pickin [110]	LTL	SDL
F49	Gibson [13]	TLA	LOTOS
F50	Plath, Ryan [111]	CTL	SMV
F51	Calder, Miller [112]	LTL	PROMELA
F52	Stepien, Logrippo [113]	LOTOS	LOTOS
F53	Capellmann, Combes, Petterson, Renard, Ruiz [114]	MSC	SDL
F54	Kamoun, Logrippo [115]	CTL	LOTOS
F55	Bousquet, Ouabdesselam, Richier, Zuanon [116, 117]	Lustre	Lustre
F56	Guelev, Ryan, Schobbens [118]	DC	SMV
F57	Thomas [119]	Temporal Logic	LOTOS
F58	Bouma, Levelt, Melisse, Middleburg, Verhaard [120]	TL	SDL
F59	Gammelgaard, Kristensen [121]	FOL	State Transition rules

Table 2.23: The Approach by Plath and Ryan

F 50: The Approach by Plath and Ryan [111]	
Notation Used	Computation Tree Logic (CTL) and Symbolic Model Verifier (SMV)
Approach Main Idea	The main idea of the approach is to describe the features formally as units of functionalities which can be understood without much knowledge of the base system. The features are integrated on top of the base system and the new extended system is verified. The verification of the new extended system includes verification of the extended system properties and verification of the extended system behaviour
Technique of the Approach	• Model the base system using the extended SMV code • Model the system properties using CTL • Verify the base system against the properties using SMV model checker • Model the features using SMV code • Integrate the features on top of the base system using the tool SFI (SMV Feature Integrator which is a tool tat the authors developed) • Verify the extended system against the set of properties described in the second set and was modeled using CTL. Detect any inconsistencies using the SMV model checker
Results	• Case study in the telecommunications domain based on the telephony features • Case study in the Lift system
Types of Interactions Detected	Logical inconsistencies

Table 2.24: The Approach by Calder and Miller

F 51: The Approach by Calder and Miller [112]	
Notation Used	Linear Temporal Logic (LTL) and PROMELA
Approach Main Idea	The main idea of the approach is to consider the base system service to develop the right level of abstraction of needed to ensure that effective reasoning techniques are established before proceeding to add features. Once this is done, features are added. The PROMELA implementation is augmented with the new feature behaviour, primarily through the use of an inline function, and then validated. Interaction detection analysis takes two forms: static analysis which is inspection of the PROMELA code, and dynamic analysis which is reasoning over combinations of sets of logical formulae and configurations of the feature
Technique of the Approach	• Model the base system as a set of properties and as a finite state automata (use LTL for properties and PROMELA for finite state automata) • Also, model the features as properties and finite state automata • Add the features on top of the base system • Perform static analysis to detect inconsistencies of the syntax of the features • Perform dynamic analysis of the model and the properties using the tool SPIN
Results	Case study in the telecommunications domain based on the telephony features
Types of Interactions Detected	• Non-determinism • Logical Inconsistencies • Violation of properties

Table 2.25: The Approach by Bousquet, Ouabdesselam, Richier, Zuanon

F 55: The Approach by Bousquet, Ouabdesselam, Richier, Zuanon [116, 117]	
Notation Used	Lustre
Approach Main Idea	The main idea of the approach is to represent the system behaviour and properties using Lustre. Features validation should be conducted in an interactive way by observing different features behaviours through sequences of exchange between the user and the telephony system executable specifications. Feature validation is done through testing to save time and resources.
Technique of the Approach	• Build a Lustre program that consists of the basic call service properties and the properties of the features • Apply testing methods that are part of Lustre to validate a specific feature F • Detect interactions between features by confronting each property of all available features to the new feature F. This is done incrementally by having a Lustre program gathers the properties of the feature to be compared and confront it with the properties of the new feature F. Several testing methods are then applied and an interaction is detected when the Lustre testing model output a false result at any time. • Perform static analysis to detect inconsistencies of the syntax of the features • Perform dynamic analysis of the model and the properties using the tool SPIN
Results	Case study in the telecommunications domain based on the telephony features
Types of Interactions Detected	Logical Inconsistencies

2.3 Surveying the Requirements Interactions Management Area

2.3.1 The requirements Interaction Management Problem

Requirements Interaction Management (RIM) was discussed in detail by Robinson *et al.* in [11] and it was defined as "the set of activities directed towards the discovery, management, and disposition of critical relationships among a set of requirements". It is very similar to feature interactions in the telecommunications domain in that they both try to detect possible interactions between features or requirements and provide guidance on how to resolve these interactions. Requirements interaction approaches are complete management approaches that include identification of interaction, proposed resolutions for the interaction, and negotiation with stakeholders for the best solution. This complete solution approach is more general than the feature interaction detection approaches as discussed in Section 1.1.

The lack of proper requirements interaction management resulted in several problems that ranged from minor inconsistencies between requirements to real life disasters like the software of the Therac-25 system, the destruction of ARIANE-5, and the A320 Warsaw airplane. In the A320 airplane disaster, an interaction between two requirements led to serious results that in turn led to the destruction of the airplane as follows: In the air, braking of an airplane is not allowed. To ensure that pilots will not accidentally engage the A320's breaking system, the software has a requirement that the breaking system is not engaged unless the wheels detect the full weight of the airplane during the landing. Another requirement of the system is that the airplane will have an efficient breaking system to ensure a safe landing. However, when a Lufthansa pilot attempted to land in

Warsaw on a wet, runway in high winds, the system did not detect the full weight of the plane on the wheels, with the following results:

"The spoilers, brakes and reverse thrust were disabled for up to 9 seconds after landing in a storm on a water logged runway, and the airplane ran off the end of the runway and into a conveniently placed earth bank, with resulting injuries and loss of life" [122]

In this section we summarize some of the work done that are relevant to the requirements interaction management area.

2.3.2 Methodology for Surveying the Requirements Interaction Approaches

In this section, different approaches for requirements interaction management are presented. A classification can be made for the different approaches based on the way they can detect interaction [11]. This classification will classify an approach into one of the following categories: Classification based, Patterns based, AI planning based, Scenario analysis based, formal model checking based, and runtime monitoring based.

However, in this section, the category formal model checking based approaches is not considered as this was considered in detail in section 2.2. Also, the runtime monitoring based approaches are not considered as they are irrelevant for this thesis.

Since the surveyed approaches are interaction management approaches, i.e., they are not concerned with only detecting interactions but have many other activities, the focus within each approach will be on the detection part as the rest of the approach is irrelevant to this thesis.

The survey presented in this section is based on the following resources:

1. An extensive survey conducted by the author using online resources on the World Wide Web and online libraries such as the IEEE [32], ACM [33], and CITESEER [34] digital libraries.
2. The survey by Robinson *et al.* [11] regarding the different approaches in the area of requirements interaction management.

2.3.3 Classification Based Approaches

The classification based approaches detect requirements interactions by comparing requirements against a-priori model of requirements interactions. The basic idea is to build a knowledge of all commonly known interactions that would occur between requirements (e.g., non-functional requirements), and then classify the requirements and compare them to the rules and knowledge that was built previously. For example, consider the two requirements R1 that can be classified to be a high accuracy requirement and R2 that can be classified into a low cost requirement. From the a-priori knowledge on non-functional requirements that the low cost is interacting with high accuracy, hence R1 and R2 are considered as interacting requirements.

Approaches that fall in this category are: WinWin approach by Boehm in 1996 [15], NFR approach by Mylopoulos *et al.* in 1992 [123], and Viewpoints by Nuseibeh *et al.* in 1994 [124].

2.3.3.1 The WinWin Approach [15]

The WinWin approach was built to support collaboration between a wide range of stakeholders with the ultimate goal of getting each stakeholder to be a winner (i.e., his needs are fulfilled). To achieve this, a-priori model on negative interactions between different non-functional requirements was built in the QARRC project [15].

Interaction identification is done when an analyst enters a new requirement R for a specific stakeholder into the database of the project under consideration. The QARRC will then classify the new requirement under one of the non-functional categories C1 (e.g., Accuracy) and starts searching for other non-functional requirements categories that would interact with C1 using the a-priori model. Once an interacting non-functional requirement category C2 is identified, all previously entered requirements that were classified under this category C2 are identified as interacting requirements with the new requirement R. The QARRC will then send a conflict advisor note to all concerned stakeholders.

The QARRC model in the WinWin approach suffers from the following problems:

- The interaction detection is based on a-priori model and hence it cannot identify new interactions that are not included in that a-priori model.
- The model is currently able to detect only non-functional requirements interactions and it does not consider technical and behavioural interactions which are in many cases the basic core of the system.
- The model uses implementation strategies for linking the non-functional requirements categories together and identifying if they interact. This means that these strategies need continuous update.

2.3.3.2 The Non-Functional Requirement (NFR) Approach [125]

The NFR approach was built to model and analyze non-functional requirements. The approach is based on building dependencies graphs for the requirements of the system (either functional or non-functional) in the form of AND/OR hierarchies. The interaction detection is done when an analyst enters a new requirement R and models it into the dependencies graphs. The first step will be to associate the new requirement with existing non-functional requirements. Association here means that the new requirement when entered will be known, via the a-priori model, to have positive or negative effect on some non-functional requirements. The second step will be to propagate the effect to all other non-functional requirements to estimate the cumulative effect of the requirement on the overall non-functional requirements.

The NFR approach suffers from the following problems:

- The NFR approach is used to target the interactions and effects with respect to only non-functional requirements
- The NFR approach is based on a-priori knowledge and hence it cannot tackle new interactions that are not included in the a-priori model
- The NFR approach is based on human experts in building the hierarchy of the requirements with AND/OR relations. However, human experts can make mistakes.
- If a link is missed between two requirements then the interaction cannot be propagated to other levels of the hierarchy.

2.3.3.3 The Viewpoint Approach [124]

Viewpoints were introduced as a means to partition requirements of different stakeholders and analyze them for conflicting views. Viewpoints address the integration of the different and heterogeneous viewpoints from stakeholders which are known to be part of the requirements engineering known problems.

The interaction detection is done by having the analyst representing the new requirement as a viewpoint. The analyst can then apply consistency rules to determine inconsistencies between the new requirement and the other requirements.

Viewpoints are expressed usually using a language which can be dataflow diagrams [126] or state transition diagrams [127]. Consistency rules are built using a formal rule pattern and are based on a priori knowledge on the different types of inconsistencies and interactions that can occur.

The viewpoints approach suffers from the following problems:

- It is aimed to detect inconsistencies rather than technical and behavioural interactions
- It partially uses formal languages such as state transition diagrams, which require heavy mathematical modeling, to represent the viewpoints in some cases.

2.3.4 Patterns Based Approaches

Pattern based approaches are approaches that detect interactions through the comparison of requirements with detection pattern conditions and interaction is found when there is a match. An interaction pattern uses pre and post conditions to constrain their use in specific situations and hence identify interactions.

An example of the approaches that can be classified as patterns based approaches is the KAOS project [128] which defines formal interaction patterns for identifying interactions.

2.3.4.1 The KAOS Approach [128]

The Knowledge Acquisition in autOmated Specification of software (KAOS) is a broad project that includes meta-modeling, specification methodology, interaction identification, learning, and reuse. We focus our efforts on the interactions detection with patterns part. The KAOS detects requirements interactions as the following types: Process level deviation, Instance level deviation, Terminology clash, Designation clash, Structure clash, Conflict, Divergence, Competition, and Obstruction.

For each one of these interaction types, the KAOS applies the corresponding interaction pattern to detect interactions under this type. For example, in the divergence interactions, apply the divergence pattern to generate boundary conditions sufficient to detect interactions. The divergence pattern is of the form:

"Given assertions of the Achieve-Avoid pattern: $(P \Rightarrow \Diamond Q) \wedge (R \Rightarrow \Box \neg S) \wedge (Q \Rightarrow S)$, consider the boundary condition: $\Diamond (P \wedge R)$"

The KAOS approach has the following problems:

- It is a heavy weight approach
- It uses formal notations (temporal logic) to define its interaction patterns
- All requirements must be represented using formal notations (temporal logic)

2.3.5 AI Planning Based Approaches

The AI planning approaches divides requirements into operational and non-operational requirements. For operational requirements, AI planning approaches use Program Slicing techniques [129] to highlight semantic differences. However, for non-operational requirements, which are presented as system goals, planning techniques can be used to detect interactions when the planner cannot find a plan for the conjunction of the requirements.

Example of approaches that fall in this category is the Deficiency-Driven Requirements Analysis (DDRA) [130].

2.3.5.1 The DDRA Approach [130]

The DDRA was designed to use AI techniques to get assistance in analyzing requirements for deficiencies. Several prototypes for achieving this goal were developed including OPIE [131] and Oz [132].

The interaction detection is done by simulation. The planner OPIE will design and simulate execution of an agent and environment that will lead to the satisfaction of a requirement or the failure of a requirement. This method was used to validate individual requirements against the environmental constraints in the system.

Another way to detect interactions between two requirements was to use OPIE to analyze the conjunction execution of two requirements held by different stakeholders. If conflict existed between the two requirements, which is usually in the form of inconsistencies between the logical formulae of the two requirements, then the planner Oz is used to identify the point at which a predicate was violated.

The DDRA approach suffers from the following problems:

- It uses formal notations for representing requirements (predicates logic)
- It is based on validating arbitrary constraints to represent the system

2.3.6 Scenario Analysis Based Approaches

Scenario analysis based approaches detect interactions by simulating a sequence of events scenario to describe some aspect of system behaviour. In scenario analysis based approaches, the aim is to check if a specific scenario can satisfy the requirements under consideration. If a scenario fails to satisfy the requirements, then there is an interaction between these requirements. Sometimes, scenario analysis is performed by model checking tools or it is performed manually by having the analyst check the outcome of the scenario to identify if an interaction exists or not.

An example of approaches that fall in this category is the Software Cost Reduction (SCR) approach [133].

2.3.6.1 The SCR Approach [133]

The SCR approach was used to specify and analyze real time embedded software systems. In SCR, the requirements are formally modeled and then a set of tools can be used to analyze the system for interactions. Two types of interactions can be detected using SCR. The first type of interactions is static interactions which includes inconsistencies and deadlocks. The second type of interactions is based on modeling the behaviour of the system with a model checker and identifying specific requirements properties that need to be checked. The SCR will analyze and detect interactions using scenarios. The output will be a trace that describes a scenario by which the requirement property under investigation fails to hold.

The SCR approach suffers from the following problems:

- SCR uses formal modeling to represent requirements and properties
- SCR uses model checking for interaction detection using scenarios which are problematic due to the state explosion problem
- SCR requires detailed design information that might not be available at the requirements level

2.4 Summary

This Chapter presented a summary of the current state of the art on approaches for detecting interactions. The survey conducted in this chapter was divided into a survey regarding the approaches in the feature interactions research community and a survey regarding the approaches developed in the requirements engineering research community. The surveys presented in this chapter are intended as the necessary background to understand the current state of the art regarding interaction detection approaches. The first survey included 7 semi-formal approaches, which were described in detail, and 59 formal approaches of which a few were described in details. The survey conducted in the requirements engineering research area included 6 approaches which were all described in detail.

CHAPTER THREE: A REQUIREMENTS INTERACTION TAXONOMY

3.1 Introduction

Requirements often interact when developing new systems because of the heterogeneity and diversity of stakeholders [124]. Hence, there is a need to have a requirements interaction taxonomy that would answer questions such as: When are two requirements considered as interacting? Why are these two requirements interacting? How do we detect this interaction? And how do we resolve it?

To the best of our knowledge, not much work has been done in the area of general requirements interaction taxonomies. Even though Robinson *et al.* [11] defined in detail the concept of requirements interactions, their work did not include in-depth information on when two requirements are considered interacting and how to detect such interactions between the two requirements. To this end, other work and research have been done and published in the area of feature interactions. In 1994, Cameron *et al.* [134] published a paper describing a benchmark for classifying the different categories of feature interactions. However, this paper is very specific to the telecommunications domain and all examples are related to interactions between telephony features and therefore it is very hard to be generalized. In 2000, Gibson *et al.* [135] presented a taxonomy for triggered interactions using fair objects semantics. This work builds on the assumption of "having a set of triggered features and using a semantic point of view for classifying interactions between those telecommunications features". Hence, Gibson *et al.'s* work [135] cannot be used beyond its assumption especially in cases where there can be triggered and non-triggered (non-functional) requirements. In 2004, Reiff-Marganiec and Turner [68] presented a taxonomy for identifying policy conflicts. However, this work focuses on the social nature of policies interactions and the social explanations of why they occur. Also, the taxonomy in [68] is geared towards policy domain and therefore not generally applicable. There are also research efforts to present partial taxonomies as sections of papers or thesis where no claim of completeness has been made [21, 136-138].

This chapter tries to address these shortcomings and presents a general interaction taxonomy for classifying and identifying requirements interactions. The proposed taxonomy can be represented in the shape of a four-layered pyramid where the first layer describes 9 main interaction categories, the second layer describes 24 interaction subcategories, the third layer describes 37 interaction types, and finally the fourth layer describes 37 interaction scenarios. Each interaction scenario has an associated interaction detection guideline. This structure addresses the lack of details that exist in other interaction taxonomies (e.g., [11, 21]). Moreover, the proposed interaction taxonomy was compared to other existing taxonomies in the literature and the obtained results were in favour of the proposed interaction taxonomy as seen in Section 3.4.

This chapter is structured as follows: Section 3.2 presents the concept of system decomposition. Section 3.3 presents the proposed requirements interaction taxonomy. Section 3.4 compares the proposed interaction taxonomy with already existing approaches. Section 3.5 presents the limitations of the proposed interaction taxonomy. Finally, section 3.6 summarizes the chapter.

3.2 System Decomposition

3.2.1 The Concept of System Decomposition

The main goal of the proposed requirements interaction taxonomy is to define interaction scenarios that fully describe interactions between requirements during the requirements engineering phase of system development. The output of the requirements engineering phase is a requirements specification document that contains a set of requirements that describe stakeholders' needs. This set of requirements can either describe certain properties that have to be preserved (static view) or dynamic behaviour which the system exhibits when certain triggers occur (dynamic view). Usually, there is also a description of the available resources that the system will use (environmental view). Therefore, we consider a system that comprises the following components:

- System Axioms: Each system axiom describes certain properties of the system that must be preserved. For example, in the lift system [139], a system axiom states that "At any time the user can press a call button to call the lift". This property must be preserved at all times to ensure the proper operation of the lift and hence it is considered a system axiom.
- Dynamic Behaviour Requirements: Each dynamic behaviour requirement describes how the system should behave when it is in a certain state and a specific trigger event occurs. For example, a requirement from the lift system [139] might state the following: "When the lift stops at floor K, it will open its doors". This dynamic behaviour requirement should perform the action "open lift doors" when the trigger event "the lift stops at floor K" occurs.

- Resources: Each resource describes physical elements that the system uses to fulfill its requirements. For example, Infra Red sensors (IR) in security systems are considered as resources used to detect motion.

The difference between a system axiom and a dynamic behaviour requirement is that the latter contains a certain action during a transition of system states when the system receives a specific trigger. On the other hand, system axioms are properties that are neither related to events nor contain transitions of system states.

3.2.2 System Representation using Attributes

A system is usually defined by textually describing all the elements of the three system components: system axioms, dynamic behaviour requirements, and resources. However, the textual description is often long, ambiguous, and easy to get lost in. For example, if it is required to examine the trigger events in a system, then a whole textual document must be read and analyzed in order to identify those trigger events. We propose the use of attributes to describe the system. In general, an attribute can be defined as a part that belongs to a bigger entity and characterizes this entity. Examples of attributes used to represent dynamic behaviour requirements include: Prestate, Trigger Event, Action, and Next State attributes. The values of these attributes for a dynamic requirement are determined from the textual description of that dynamic behaviour requirements.

This concept of representation using attributes can be applied to all the three system components as follows:

Consider a system S which can be described using the following equation:

$$S = \Omega \cup D \cup \Psi \qquad (3.1)$$

where S: the system under consideration, Ω: system axioms component

D: dynamic behaviour requirements component, Ψ: resources component

Mathematically, Ω is defined by:

$$\Omega = \{\omega_1, \omega_2, \ldots, \omega_m\} \qquad (3.2)$$

where $\omega_1 .. \omega_m$ are all the system axioms in the system

Every individual system axiom ω_i can be represented using system axioms attributes. Based on the different textual formats that system axioms can have, our research found that any system axiom can be represented by:

$$\omega_i = < \Upsilon_{i1}, \Upsilon_{i2}, \Upsilon_{i3}, \Upsilon_{i4}, \Upsilon_{i5}, \Upsilon_{i6} > \qquad (3.3)$$

where Υ_{ij}: is the jth attribute associated with the ith system axiom, j=1, 2, …, 6

System axiom ω_i can be defined using the attributes $\Upsilon_{i1} \ldots \Upsilon_{i6}$ as follows:

Υ_{i1}	ID: A unique ID number corresponding to the system axiom number given in the requirements document
Υ_{i2}	Description: An informal description of the system axiom as specified in the requirements document
Υ_{i3}	Rule: A description of the required property encapsulated in this system axiom that must be preserved
Υ_{i4}	Condition: A description of any specific conditions on preserving the property described in the Rule attribute
Υ_{i5}	Parameters: A description of any parameters that are listed in the system axiom body {optional}
Υ_{i6}	Parameter range: A description of any restrictions on the values that the parameters can have {optional}

The dynamic behaviour requirements component can be mathematically represented by: $D = \{d_1, d_2, \dots, d_n\}$ (3.4)

where d_i: represents the i^{th} dynamic behaviour requirement in the system

The i^{th} dynamic behaviour requirement d_i can be represented using dynamic behaviour requirements attributes. After a study of the different possible textual representation that a dynamic behaviour requirement can take, it was found that any dynamic behaviour requirement can be represented using 8 attributes:

$$d_i = < \Gamma_{i1}, \Gamma_{i2}, \Gamma_{i3}, \Gamma_{i4}, \Gamma_{i5}, \Gamma_{i6}, \Gamma_{i7}, \Gamma_{i8} > \qquad (3.5)$$

where Γ_{ij}: is the jth attribute associated with ith dynamic requirement, j=1, 2, …,8

Dynamic behaviour requirement d_i can be defined using $\Gamma_{i1} \dots \Gamma_{i8}$ and written as follows:

Γ_{i1}	ID: A unique ID number corresponding to the dynamic behaviour requirement number in the requirements document
Γ_{i2}	Description: Informal description of the dynamic behaviour requirement given in the requirements document
Γ_{i3}	Pre-state: A description of the required system state prior to the execution of this dynamic behaviour requirement
Γ_{i4}	Trigger event: A description of the trigger event required for this dynamic behaviour requirement to execute
Γ_{i5}	Action: A description of the action carried out by this dynamic behaviour requirement once triggered
Γ_{i6}	Next state: A description of the next state that the system should reach once this dynamic behaviour requirement finishes executing
Γ_{i7}	Parameters: A description of any parameters that are listed in the dynamic behaviour requirement body {optional}
Γ_{i8}	Parameter range: A description of any restrictions on the values that the parameters can have {optional}

Finally, consider Ψ as the resources component within a system. It is defined by:

$$\Psi = \{\psi_1, \psi_2, \ldots, \psi_k\} \tag{3.6}$$

where ψ_i: denotes the i^{th} resource in the system

Resources attributes can be used to represent a resource ψ_i. Based on a study of the different possible textual representations that a resource can take 5 different attributes have been identified to represent a resource:

$$\psi_i = < \Lambda_{i1}, \Lambda_{i2}, \Lambda_{i3}, \Lambda_{i4}, \Lambda_{i5} > \tag{3.7}$$

where Λ_{ij}: is the jth attribute associated with ith resource, j=1, 2, …, 5

Resource ψ_i can be defined using the attributes $\Lambda_{i1} \ldots \Lambda_{i5}$ as follows:

Λ_{i1}	ID: A unique ID number corresponding to the dynamic behaviour requirement number given in the requirements document
Λ_{i2}	Description: An informal description of the dynamic behaviour requirement specified in the requirements document
Λ_{i3}	Availability: A description of this resource's availability constraints {optional}
Λ_{i4}	Performance: A description of this resource's performance constraints {optional}
Λ_{i5}	Interface: A description of this resource's interface constraints {optional}

It is worth mentioning that an optional attribute, which is labelled by {optional}, will only have a value if textual description of the system axiom, dynamic behaviour requirement, or resource describes these attributes. For example, if the resource ψ_i textual description has a certain constraint value on its availability value, then the attribute Availability of ψ_i is assigned to this availability value.

3.3 The proposed Interaction Taxonomy

3.3.1 General Architecture

In order to address the problem of requirements interaction in software systems, many questions arise such as:

- WHERE can interactions occur in a system? Interactions can occur between two elements from two different components, e.g., a system axiom from the system axioms component and a resource from the resources component. Alternatively, interactions can occur between two elements within one component, e.g., system axiom A and system axiom B from the system axioms component.
- WHAT attributes cause the interaction? This requires the identification of those attributes that cause the interaction to really occur.
- WHY does the interaction occur between the attributes? This question looks for the reasons of why the attributes are interacting.
- HOW can the interactions be identified? This question looks into how the different types of requirements interactions can be detected.

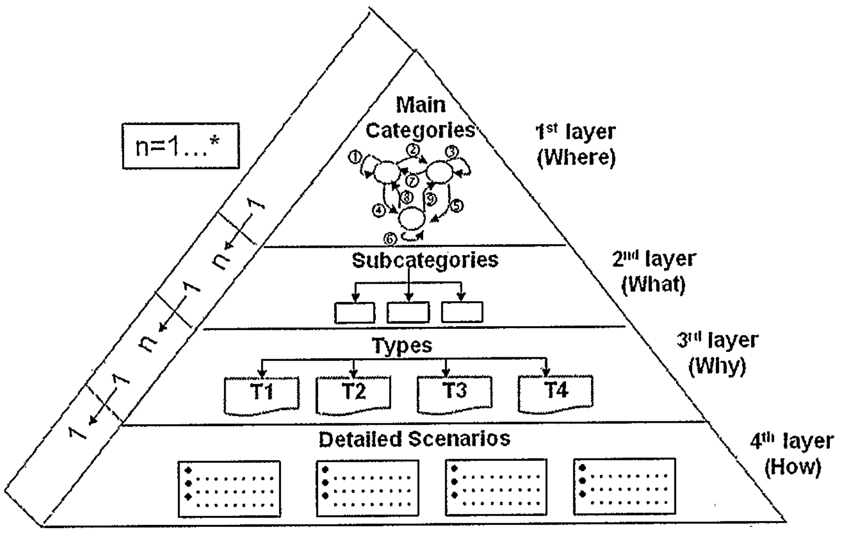


Figure 3.1: General architecture of the proposed interaction taxonomy

The question on how to resolve interactions was left out of the taxonomy because the focus of this thesis is only on the detection of interactions. Moreover, different resolutions can heavily vary according to stakeholders' preferences.

The architecture of the proposed interaction taxonomy addresses the questions listed above in a gradual manner as shown in Figure 3.1. The proposed taxonomy starts in the first layer by addressing the question of WHERE in the system interactions can occur. Whenever two elements (either from two different components, e.g., a system axiom and a resource, or from the same component, e.g. two system axioms) are interacting, they are said to form a main interaction category.

The second layer of the taxonomy addresses the question of WHAT attributes of the two system elements, identified in the first layer, cause the interaction. The second layer contains interaction subcategories. Each interaction subcategory describes the two attributes that cause the interaction.

The third layer of the proposed taxonomy addresses the question of WHY the attributes, identified in the second layer, are interacting. The third layer contains interaction types with each interaction type describing why the two attributes from the second layer are interacting.

The fourth layer of the proposed interaction taxonomy addresses the question of HOW to detect interaction types, identified in the third layer, in any software system. This layer contains interaction scenarios where each scenario is used to describe in detail a specific interaction type and how to detect it.

The elements of the first and the second layers of the proposed interaction taxonomy have a 1:n relationship. This means that each main interaction category in the first layer can have up to n (where $n \geq 1$) interaction subcategories in the second layer depending on what attributes cause interactions.

The elements of the second and the third layers of the proposed interaction taxonomy also have a 1:n relationship. This means that each interaction subcategory in the second layer can have up to n (where $n \geq 1$) interaction types in the third layer depending on why the two attributes in the interaction subcategory are interacting.

The elements of the third and fourth layers of the proposed interaction taxonomy have a 1:1 relationship. This means that each interaction type in the third layer will have only one corresponding interaction scenario in the fourth layer.

3.3.2 First Layer: Main Interaction Categories

Any two elements (out of system axioms, dynamic behaviour requirements, and resources) that interact are said to form a main interaction category (whether these two elements are from two different components or from the same component).

The number of the main interaction categories is 9 as shown in Figure 3.2 and are listed as follows:

① Two interacting system axioms.

② A system axiom interacting with a dynamic behaviour requirement.

③ Two interacting dynamic behaviour requirements.

④ A system axiom interacting with a resource.

⑤ A dynamic behaviour requirement interacting with a resource.

⑥ Two interacting resources.

⑦ A dynamic behaviour requirement interacting with a system axiom.

⑧ A resource interacting with a system axiom

⑨ A resource interacting with a dynamic behaviour requirement

In the remainder of this chapter we will focus our efforts on explaining and describing only the main interaction category "Two Interacting Dynamic Behaviour Requirements" as an ongoing example. However, the rest of the taxonomy will be listed in Appendix B.

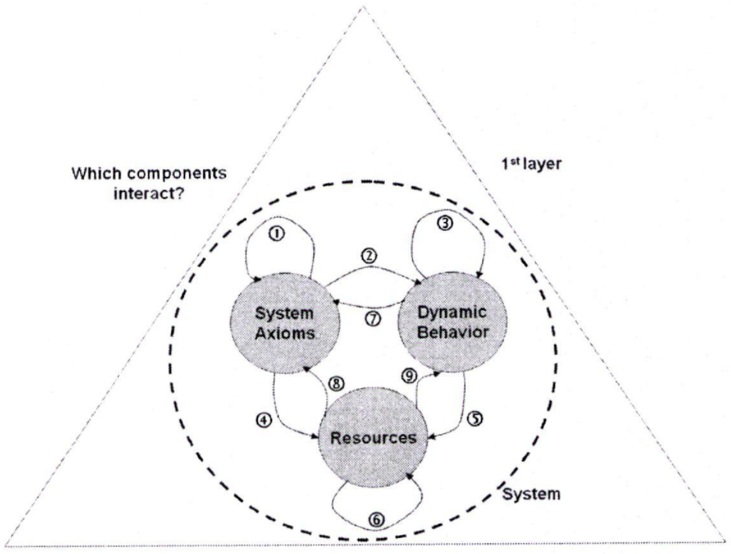


Figure 3.2: First layer of the proposed interaction taxonomy

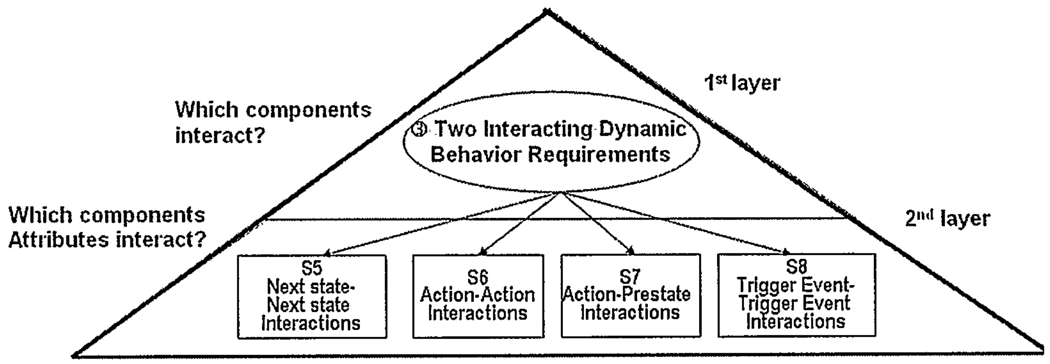

1st layer
Which components interact?
③ Two Interacting Dynamic Behavior Requirements
Which components Attributes interact?
2nd layer
S5 Next state-Next state Interactions
S6 Action-Action Interactions
S7 Action-Prestate Interactions
S8 Trigger Event-Trigger Event Interactions

Figure 3.3: Second layer of the proposed interaction taxonomy

3.3.3 Second Layer: Interaction Subcategories

The second layer of the proposed interaction taxonomy contains interaction subcategories that are linked to the first layer through an attribute-based decomposition. An interaction subcategory describes what attributes of the two interacting system elements, identified in the first layer, cause the interaction. Therefore, to generate the second layer interaction subcategories, each possible pair of attributes between the two interacting elements is first listed, with the first attribute is from the first element and the second attribute is from the second element. Then the obtained pairs of attributes are analyzed to determine which ones can cause interactions. Any pair of attributes that could cause an interaction situation is then listed and considered to be an interaction subcategory.

Therefore, the main interaction categories from the first layer will be decomposed into different numbers of interaction subcategories in the second layer depending on the outcome of the analysis of attributes pairs (e.g., as seen in Figure 3.2, the main interaction category ③ has four subcategories S5-S8, whereas as seen in Appendix B.3 the main interaction category ④ has three subcategories S9-S11).

The first layer's 9 main interactions categories resulted in the following 24 interaction subcategories in the second layer: 1 subcategory (S1) from main category ①, 3 subcategories (S2, S3, and S4) from main category ②, 4 subcategories (S5, S6, S7, and S8) from main category ③, 3 subcategories (S9, S10, and S11) from main category ④, 3 subcategories (S12, S13, and S14) from main category ⑤, 3 subcategories (S15, S16, and S17) from main category ⑥, 1 subcategory (S18) from main category ⑦, 3 subcategories (S19, S20, and S21) from main category ⑧, and 3 subcategories (S22, S23, and S24) from main category ⑨.

We continue with the ongoing example of presenting and explaining the subcategories of interactions derived from the main interaction category ③ "Two interacting Dynamic Behaviour Requirements". The remaining interaction subcategories associated with the other eight main interaction categories, are presented in Appendix B.

Figure 3.3 shows how the third main interaction category ③ from the first layer is decomposed into 4 interaction subcategories in the second layer. The decomposition was based on the attributes of dynamic behaviour requirements, namely: Prestate, Trigger event, Action, and Next state (refer to Section 3.2.2). The other two attributes "Parameters" and "Parameters range", described in section 3.2.2 as part of the dynamic behaviour requirement set of attributes, are not used in the decomposition. This is because these two attributes will not cause interaction situations with other attributes but they are used to show the effects that parameters can have on the interaction and how the parameters values can heavily affect the interaction. This is further explained in the fourth layer of the proposed interaction taxonomy.

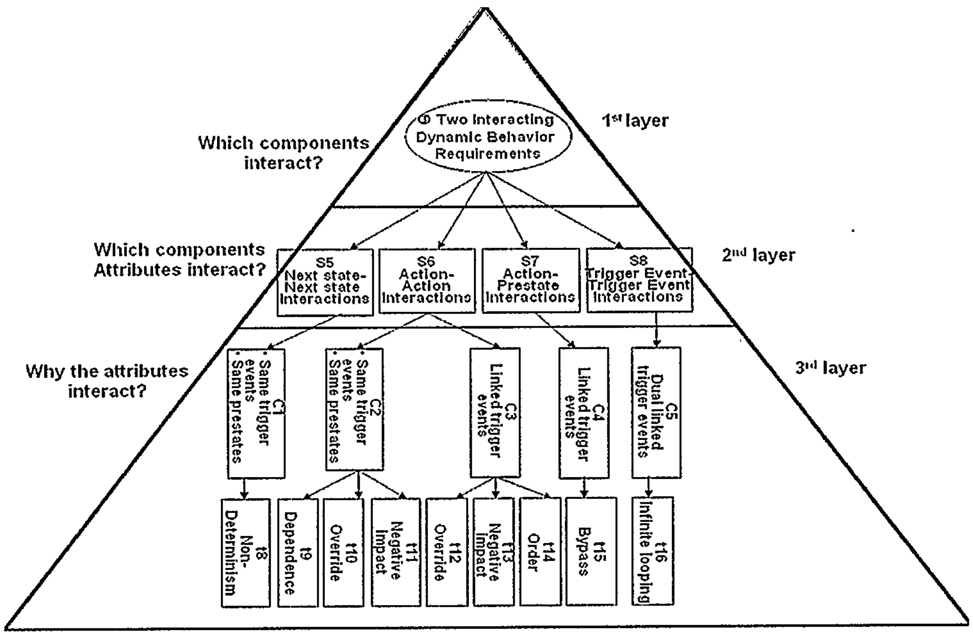
After analyzing the possible pairs of attributes that can form interaction categories, only four pairs were found to really represent interactions subcategories as follows:

- **S5: Next State–Next State interactions:** This subcategory contains all the interactions that arise between two dynamic behaviour requirements because the next state attribute of the first requirement interacts with the next state attribute of the second requirement.
- **S6: Action–Action interactions:** This subcategory contains all interactions that arise between two dynamic behaviour requirements because the action attribute of the first requirement interacts with the action attribute of the second requirement.
- **S7: Action–Prestate interactions:** This is a subcategory that contains all the interactions that arise between two dynamic behaviour requirements because the action attribute of the first requirement interacts with the prestate attribute of the second requirement.
- **S8: Trigger Event–Trigger Event interactions:** This is a subcategory that contains all the interactions that arise between two dynamic behaviour requirements because the trigger event attribute of the first requirement interacts with the trigger event attribute of the second requirement.

Note that the numbering started from 5 because there are other 4 subcategories derived from the first two main interaction categories ① and ②.

3.3.4 Third Layer: Interaction Types

The third layer of the interaction taxonomy describes the reasons why the attributes, identified in the interaction subcategories in the second layer, are interacting. Each one of these reasons forms an interaction type. Therefore, the number of interaction types for an interaction subcategory will depend on the number of reasons that can cause the two attributes of this interaction subcategory to interact.



③ Two Interacting Dynamic Behavior Requirements
1st layer
Which components interact?
Which components Attributes interact?
S5 Next state-Next state Interactions
S6 Action-Action Interactions
S7 Action-Prestate Interactions
S8 Trigger Event-Trigger Event Interactions
2nd layer
Why the attributes interact?
C1 • Same trigger events • Same prestates
C2 • Same trigger events • Same prestates
C3 Linked trigger events
C4 Linked trigger events
C5 Dual linked trigger events
3rd layer
t8 Non-Determinism
t9 Dependence
t10 Override
t11 Negative Impact
t12 Override
t13 Negative Impact
t14 Order
t15 Bypass
t16 Infinite looping

Figure 3.4: Third layer of the proposed taxonomy

Sometimes there are certain constraints on an interaction type to occur. For example, consider the interaction subcategory S5 "Next State–Next State interactions" derived from the main interaction category ③ "Two interacting dynamic behaviour requirements" (see Figure3.4).

This interaction subcategory has only one interaction type t8 called "Non-Determinism" in the third layer that describes that the attribute Next State of the first requirement interacts with the attribute Next State of the second requirement because they have different values and will therefore cause a non-determinism situation in the system.

However, for this interaction type to occur, the two dynamic behaviour requirements must execute simultaneously, i.e. they must have: (same prestates) AND (same trigger events). This is considered to be a constraint on the interaction type "Non-Determinism" and therefore the subcategory "Next State–Next State interactions" is connected to the "Non-Determinism" interaction type through the constraint C1 "Same prestates AND same trigger events".

It must be noted that some interaction types can be repeated more than once under the same subcategory because this interaction type occurs under two different constraints (e.g., t10 and t12 under S6 in Figure 3.4).

Overall, the 24 interaction subcategories from the second layer resulted in 37 interaction types and 5 constraints in the third layer as shown in Table 3.1.

We continue with our ongoing example and describe only types of interactions that are derived from the subcategories S5 "Next State – Next State interactions", S6 "Action – Action interactions", S7 "Action-Prestate interactions", and S8 "Trigger Event – Trigger Event interactions", which are presented in section 3.3.3 as subcategories derived from main interaction category ③ "Two interacting Dynamic Behaviour Requirements". The remaining types of other interaction subcategories are presented in Appendix B.

Table 3.1: Summary of the resulting interaction types in the third layer

S1	t:2, C:0 (Appendix B.1)	**S2**	t:2, C:0 (Appendix B.2)	**S3**	t:1, C:0 (Appendix B.2)
S4	t:2, C:0 (Appendix B.2)	**S5**	t:1, C:1 (Figure 3.3)	**S6**	T:6, C:2 (Figure 3.3)
S7	t:1, C:1 (Figure 3.3)	**S8**	t:1, C:1 (Figure 3.3)	**S9**	t:2, C:0 (Appendix B.3)
S10	t:1, C:0 (Appendix B.3)	**S11**	t:2, C:0 (Appendix B.3)	**S12**	t:2, C:0 (Appendix B.4)
S13	t:1, C:0 (Appendix B.4)	**S14**	t:2, C:0 (Appendix B.4)	**S15**	t:1, C:0 (Appendix B.5)
S16	t:1, C:0 (Appendix B.5)	**S17**	t:1, C:0 (Appendix B.5)	**S18**	t:2, C:0 (Appendix B.6)
S19	t:1, C:0 (Appendix B.7)	**S20**	t:1, C:0 (Appendix B.7)	**S21**	t:1, C:0 (Appendix B.7)
S22	t:1, C:0 (Appendix B.8)	**S23**	t:1, C:0 (Appendix B.8)	**S24**	t:1, C:0 (Appendix B.8)
<u>**Where**</u> **Si**: The ith Interaction subcategory in the second layer **t**= Number of interaction type in third layer resulting from the corresponding Si **C**= Number of constraints in the third layer resulting from the corresponding Si					

Figure 3.4 shows how the subcategories S5, S6, S7, and S8 from the second layer are associated with interaction types and constraints in the third layer of the proposed taxonomy. The details of these interactions types in the third layer are as follows (in all interaction types t8 to t16 described below, consider R1 and R2 to be dynamic behaviour requirements):

t8: "Non-Determinism" interaction type (under constraint C1): Consider R1 and R2 to have the same trigger events and the same prestates and hence will be executed together. Also consider R1 and R2 to have different values for their Next State attributes. If these two requirements are executed at the same time then the system will face an ambiguous situation in which the system is unable to determine which state to go to (the next state specified in R1 or the next state specified in R2).

t9: "Dependence" interaction type (under constraint C2): Consider R1 and R2 to have the same trigger events and the same prestates and hence will be executed together. Now, suppose that the action of R1 requires that the action of R2 be successfully executed. This means that the action of R1 depends on the action of R2, i.e., an interaction occurs if the action of R2 is not completed successfully for any reason.

t10: "Override" interaction type (under constraint C2): Consider the two dynamic behaviour requirements R1 and R2 to have the same trigger event and the same prestate and that they have been triggered and are executing simultaneously. Suppose that the action of R1 interrupts and cancels the action of R2 before its completion which means that the action of R1 has overridden the action of R2. Hence there is a negative relationship from R1 on R2, and by definition, R1 and R2 interact.

t11: "Negative Impact" interaction type (under constraint C2): Consider R1 and R2 to have the same trigger events and the same prestates and that they have been triggered and are executing simultaneously. Now suppose that the action of R1 negatively impacts the action of R2. Hence, R1 interacts with according to the interaction type t11. This interaction type is similar to t10, however, the difference is that in t10 the action of R1 will completely cancel the action of R2 while in t11 the action of R1 will only negatively impact, but not completely cancel, the action of R2.

t12: "Override" interaction type (under the constraint C3): Consider R1 and R2 to have different, but linked trigger events, i.e., the occurrence of the first trigger event is followed after some time by the occurrence of the second trigger event (Section 4.3.5 provides complete details and definition of linked events). Hence R1 and R2 are still sequentially related and prone to interactions. Suppose that R1 is triggered and starts executing. R2 is also triggered and starts executing after some time because the trigger event of R2 is linked to the trigger event of R1. Now, assume that the action of R1 is not yet completed while R2 is triggered. If the action of R2 cancels and overrides the action of R1 before its completion then there is an interaction between the two requirements. The interaction type t12 is also possible when the action of R1 overrides and cancels the action of R2. In both cases, R1 and R2 interact.

t13: "Negative Impact" interaction type (under the constraint C3): Assume R1 and R2 to have linked trigger events and hence if R1 is triggered and starts executing then R2 will also be triggered and starts executing after some time. Now, if the action of R1 negatively affects the action of R2 then the R1 interacts with R2. This interaction type can also occur when the action of R2 negatively impacts the action of R1.

t14: "Order" interactions type (under constraint C3): Consider R1 and R2 to have linked trigger events. Assume that the trigger of the first requirement leads to the trigger of the second requirement, i.e. T1~>T2. In this case, the first requirement R1 executes first then followed by the execution of the second requirement R2. Consider B1 to be the system specific behaviour after the two requirements have executed their actions. Now if behaviour B1 is different from the behaviour that the system would exhibit if R2 had started first followed by R1, i.e., T2~>T1, then there is an interaction between the two requirements. This is because the actions of the two requirements are not independent but have an effect on each other. If they were independent then the same behaviour would have been obtained no matter which action started first.

t15: "Bypass" interaction type (under constraint C4): Consider R1 and R2 with linked trigger events. Assume that R1 is triggered and starts executing and that the action of R1 bypasses the system from being in a specific state. Suppose that this specific state is the same state specified in the prestate attribute of R2. Hence when the trigger event of R2 occurs, R2 will never execute because the system is in a state different from R2's prestate. Thus R1 prevented the system from executing R2.

t16: "Infinite Looping" interaction type (under constraint C5): If R1 is triggered and starts executing such that its action will create the trigger event for R2 and hence R2 starts executing its action. Now if the action of the second requirement R2 causes the creation of the trigger event of the first requirement R1, then R1 is triggered again and starts executing its action which will again create the trigger event of R2 and so on. Therefore, R1 and R2 are forced into infinite looping and interact.

3.3.5 Fourth Layer: Interaction Scenarios

The fourth layer of the proposed interaction taxonomy contains interaction scenarios that provide details on the different interaction types by giving:

- a guideline on how to detect this type of interaction
- an example of each interaction type in a real system
- an explanation of how parameters can affect this type of interaction

The third bullet in the above list, "parameters effect", was introduced to emphasize the effect that parameters can have on the interaction between two requirements. In section 3.3.3, it was stated that the two attributes "Parameters" and "Parameters range" are used to describe the effect parameters can have on the cause and resolution of requirements interactions.

When a requirement has parameters in its body then it is called a parameterized requirement where these parameters can be assigned specific values during later system development stages. For example, a parameterized requirement from the telephony domain might state that "The phone can dial a number using X techniques." The parameter X in this requirement can take several values such as "pressing numbers on the keypad" and/or "speed dial" and/or "voice dialling".

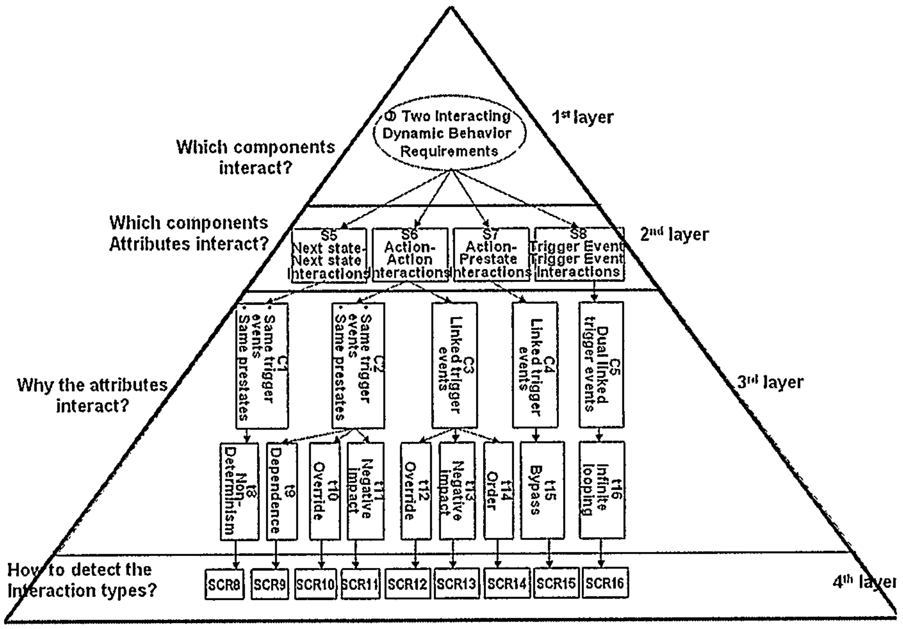

Two Interacting Dynamic Behavior Requirements
1st layer
Which components interact?
Which components Attributes interact?
S5 Next state-Next state Interactions
S6 Action-Action Interactions
S7 Action-Prestate Interactions
S8 Trigger Event-Trigger Event Interactions
2nd layer
Why the attributes interact?
C1 • Same trigger events • Same prestates
C2 • Same trigger events • Same prestates
C3 Linked trigger events
C4 Linked trigger events
C5 Dual linked trigger events
3rd layer
t8 Non-Determinism
t9 Dependence
t10 Override
t11 Negative impact
t12 Override
t13 Negative impact
t14 Order
t15 Bypass
t16 Infinite looping
How to detect the Interaction types?
SCR8
SCR9
SCR10
SCR11
SCR12
SCR13
SCR14
SCR15
SCR16
4th layer

Figure 3.5: Fourth layer of the proposed interaction taxonomy

A general template for each interaction scenario is used as a way of presenting it in a more organized manner with the following columns:

- Scenario ID: This is a unique ID that distinguishes one interaction scenario from another.
- Interaction Type: This describes the interaction type the scenario is associated with. The description does not only include the interaction type as a single leaf but it includes the whole branch starting from the main interaction category in the first layer.
- Detection Guideline: This column describes how to detect the interaction type described in the scenario by a non-expert. The detection guideline includes a textual description and, where appropriate, a graphical description.

- Example: This is an example that explains the occurrence of the interaction type, associated with this scenario, taken from a real life system.
- Parameters Effect: This column gives an example of how parameters in parameterized requirements affect the interaction described in the scenario.

Each interaction type from the third layer is associated with only one interaction scenario in the fourth layer. Hence, the fourth layer of the proposed interaction taxonomy contains 37 interaction scenarios. In the remainder of this section we continue our ongoing example and present only interaction scenarios associated with interaction types t8 to t16 as shown in Figure 3.5. The details of SCR8 to SCR16 are presented in Tables 3.2 to 3.10, respectively. The following symbols have been used:

- Ti: Trigger event of the requirement Ri
- Pi: Prestate of the requirement Ri
- Ni: Next state of the requirement Ri
- Ai: Action of the requirement Ri

It is worth mentioning that all examples presented in SCR8 to SCR 16 are taken from the smart homes domain which is described in more detail in Chapter 8.

Table 3.2: Description of the interaction scenario SCR8

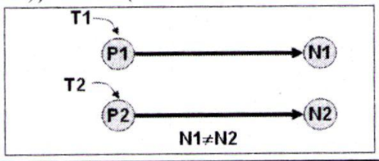
Scenario ID	SCR8
Type of Interaction	TwoInteractingDynamicBehaviourRequirements → NextState-NextStateInteractions → Non-Determinism $\|_{C1=SameTriggerEvents\&SamePreStates}$
Detection Guideline	IF {(R1.TriggerEvent=R2.TriggerEvent) AND (R1.PreState=R2.PreState) AND (R1.NextState ≠ R2.NextState)} THEN {R1 interacts with R2 under the interaction type t8} 
Example	• R1:"Adjust the audio level of the device (X1=TV) to (X2=35% of the max. volume) when the device is first turned on". • R2:"Adjust the audio level of the TV when it is turned on to the last used audio level setting before the last shutdown". • Interaction: Assume that someone was previously watching TV and has manually adjusted the TV audio level to 20% of its max volume before he shuts it down. Later on, when the TV is first turned on then both R1 and R2 are triggered at the same time. Since the audio level specified in R2 (20% of max audio level) is different from the audio level specified in R1 (35% of the max. volume), then the system will face a non-determinism situation on which state it should transit to. Should it transit to the state where the volume of TV is 20% as specified by R2 or should it transit to the state where the volume of TV is 35% as specified by R1?
Parameters Effect	If X1 was set to another audio device then there is no interaction between the two requirements R1 and R2. Moreover, if X2 is set to automatically obtain the last stored audio settings of the audio device, then there is no interaction also.

Table 3.3: Description of the interaction scenario SCR9

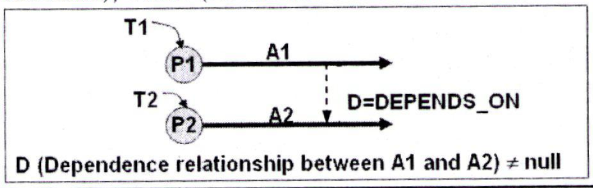
Scenario ID	SCR9
Type of Interaction	TwoInteractingDynamicBehaviourRequirements →Action-ActionInteractions → Dependence $\|_{C2=SameTriggerEvents\&SamePreStates}$
Detection Guideline	IF {(R1.TriggerEvent=R2.TriggerEvent) AND (R1.PreState=R2.PreState) AND (R1.Action DEPENDS_ON R2.Action)} THEN {R1 interacts with R2 under the interaction type t9}  T1 P1 A1 T2 P2 A2 D=DEPENDS_ON D (Dependence relationship between A1 and A2) ≠ null
Example	• R3:"Increase the temperature inside the house to the preset temperature (X3=22) when temperature reading from thermostat is ≤ {(X3=22) – 2} degrees" • R4:"Open the ventilation grills in locations (X4={LivingRoom, BedRoom1}) to allow air flow when the temperature reading from the thermostat is ≤ {(X3=22) – 2} degrees" • Interaction: When the temperature drops below 20, then both requirements R3 and R4 trigger at the same time. However the action of R3 depends on the action of R4 as the temperature is increased by pumping hot air through the ventilation grills. If R4 fails to execute for any reason, then R3 will not be able to perform its action. Even more, if the action of R4 opens only one or two ventilation grills, then the action of R3 is affected by the few opened ventilation grills and it will not be effective enough
Parameters Effect	X4 has an effect on the type of interaction between R3 and R4 as X4 determines which ventilation grills are opened. If X4 was an empty set, i.e., no ventilation grills were opened then R4 will fail to increase the temperature of the house. But if X4 was properly assigned then the dependence relationship is reduced to malfunctions situations of the ventilation grills.

Table 3.4: Description of the interaction scenario SCR10

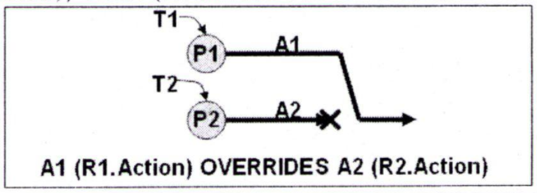
Scenario ID	SCR10
Type of Interaction	TwoInteractingDynamicBehaviourRequirements → Action-ActionInteractions → Override $\|_{C2=SameTriggerEvents\&SamePreStates}$
Detection Guideline	IF {(R1.TriggerEvent=R2.TriggerEvent) AND (R1.PreState=R2.PreState) AND (R1.Action OVERRIDES R2.Action)} THEN {R1 interacts with R2 under the interaction type t10}  T1 P1 A1 T2 P2 A2 A1 (R1.Action) OVERRIDES A2 (R2.Action)
Example	• R5:"As a security measure, secure the doors and windows of a house by having them closed starting at time (X5=11:00 pm) for (X6=6 hours)" • R6:" automatically opens the windows in (X7=LivingRoom) at time (X8=11:00 pm)" • Interaction: When the time is 11:00 pm the two requirements, R5 and R6, are triggered and start executing. However, R6 tries to open the windows but R5, which is a security requirement, will override the action of R6 and will not allow it to open the windows.
Parameters Effect	If X8 was set out the range in which R5 is active, i.e., X5 to X5+X6, then there is no interaction as R6 can execute normally. Moreover, if X6 is set to 0 hours then the user is technically disabling R5 and there is no interaction

Table 3.5: Description of the interaction scenario SCR11

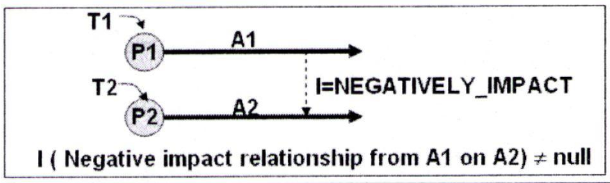
Scenario ID	SCR11
Type of Interaction	TwoInteractingDynamicBehaviourRequirements → Action -ActionInteractions → NegativeImpact $\|_{C2=SameTriggerEvents\&SamePreStates}$
Detection Guideline	IF {(R1.TriggerEvent=R2.TriggerEvent) AND (R1.PreState=R2.PreState) AND (R1.Action NEGATIVELY_IMPACTS R2.Action)} THEN {R1 interacts with R2 under the interaction type t11}  T1 P1 A1 T2 P2 A2 I=NEGATIVELY_IMPACT I (Negative impact relationship from A1 on A2) ≠ null
Example	• R6 (revisited from SCR10):" automatically open the windows in (X7=LivingRoom) at time (X8=11:00 pm)" • R7:"Increase/Decrease the temperature of the house to the temperature (X9=22) at time (X10=11:00 pm). • Interaction: When the time is 11:00 pm the two requirements are triggered and both of them start executing. Now, if the temperature outside the house is too cold or too hot then the action of R6 will negatively affect the action of R7 as R7 will try to increase/decrease the temperature of the house when the windows are opened.
Parameters Effect	If X10 was set to be a different time prior to X8 then there is no interaction as the two requirements will execute at different times. Moreover, if X7 is an empty set then there are no windows to be opened and the interaction is eliminated.

Table 3.6: Description of the interaction scenario SCR12

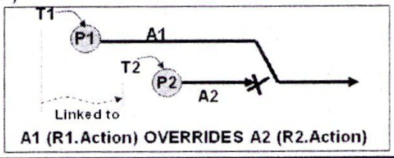
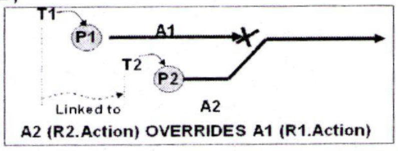
Scenario ID	SCR12	
Type of Interaction	TwoInteractingDynamicBehaviourRequirements → Action-ActionInteractions → Override$|_{C3=LinkedTriggerEvents}$	
Detection Guideline	② IF {(R1.TriggerEvent ~> R2.TriggerEvent) AND (R1.Action OVERRIDES R2.Action)} Then { R1 interacts with R2 under the interaction type t12}  T1 P1 A1 T2 P2 A2 Linked to A1 (R1.Action) OVERRIDES A2 (R2.Action)	① IF {(R1.TriggerEvent ~> R2.TriggerEvent) AND (R2.Action OVERRIDES R1.Action)} Then {R1 interacts with R2 under the interaction type t12}  T1 P1 A1 T2 P2 Linked to A2 A2 (R2.Action) OVERRIDES A1 (R1.Action)
Example	• R3 (revisited from SCR8): "Use the last stored audio level settings of the TV to adjust its volume when the TV is first turned on" • R8: "Completely shutdown power supply to all Audio/video devices starting at (X11=midnight) for (X12=5 hours)" • Interaction: Suppose that the TV was turned on just a few seconds before midnight. According to R2 the system will obtain the last stored audio level settings of the TV and starts adjusting its volume. But at midnight R8 starts executing and hence all audio/video devices including the TV are shutdown. Hence the action of R8 has overridden the action of R2 before its completion. This example is shown as the detection guideline number 2.	
Parameters Effect	If the parameter X12 was set to 0 hours, then there is no interaction as R8 will not power off any devices.	

Table 3.7: Description of the interaction scenario SCR13

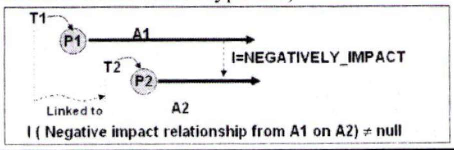
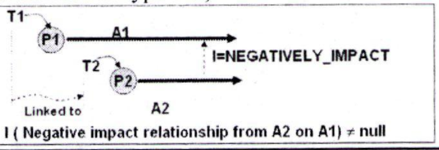
Scenario ID	SCR13	
Type of Interaction	TwoInteractingDynamicBehaviourRequirements → Action-ActionInteractions → NegativeImpact$|_{C3=LinkedTriggerEvents}$	
Detection Guideline	① IF {(R1.TriggerEvent ~> R2.TriggerEvent) AND (R1.Action NEGATIVELY_IMPACT R2.Action)} Then { R1 interacts with R2 under the interaction type t13}  T1 P1 A1 I=NEGATIVELY_IMPACT T2 P2 Linked to A2 I (Negative impact relationship from A1 on A2) ≠ null	② IF {(R1.TriggerEvent ~> R2.TriggerEvent) AND (R2.Action NEGATIVELY_IMPACT R1.Action)} Then { R1 interacts with R2 under the interaction type t13}  T1 P1 A1 I=NEGATIVELY_IMPACT T2 P2 Linked to A2 I (Negative impact relationship from A2 on A1) ≠ null
Example	• R6 (revisited from SCR10):" automatically opens the windows in (X7=LivingRoom) at time (X8=11:10 pm)" • R7 (revisited from SCR11):"Increase/Decrease the temperature of the house to the temperature (X9=22) starting at time (X10=11:00 pm). • Interaction: When the time is 11:00 pm, R7 is triggered and starts executing. Now, R7 tries to increase/decrease the house temperature to the value specified in X9 which is 22 degrees. However this needs some time and meanwhile the time gets to 11:10 pm which triggers R6. Now R6 opens the windows and consequently negatively affecting the action of R7. This example is shown as the detection guideline number 2 in the previous row.	
Parameters Effect	The parameters effect is the same as explained in SCR11. But the example above shows that when X8 had a different value, the type of interaction change from Negative Impact with same trigger event and same prestates to Negative impact with linked trigger events.	

Table 3.8: Description of the interaction scenario SCR14

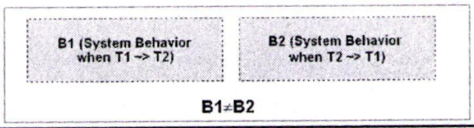
Scenario ID	SCR14
Type of Interaction	TwoInteractingDynamicBehaviourRequirements → Action-ActionInteractions → Order $\|_{C3=LinkedTriggerEvents}$
Detection Guideline	IF {(SYSTEM_BEHAVIOUR$\|_{R1.TriggerEvent \rightsquigarrow R2.TriggerEvent}$) ≠ (SYSTEM_BEHAVIOUR$\|_{R2.TriggerEvent \rightsquigarrow R1.TriggerEvent}$) } Then { R1 interacts with R2 under the interaction type t14}  B1 (System Behavior when T1 -> T2) B2 (System Behavior when T2 -> T1) B1≠B2
Example	• R9: "The system shall support a one-click remote control 911 emergency service that calls emergency centre and provides the home address and a pre-recorded message once a connection is established" • R10: "The system shall provide a regular telephone line with the set of telephony features (X13=Three Way Calling). Interaction: Suppose that an elderly resident A faces an emergency health condition (e.g. heart attack). A calls his son on the phone to take him to the hospital but meanwhile, the condition gets worst so he uses R9 to call 911. Now R9 finds the line is busy and it cannot execute 911 directly, so the system uses the Three Way Calling feature in R10 to put the son on hold and then connects to the emergency centre using 911. Consider this as the system behaviour B1 when Three Way Calling is activated first then 911 is followed later. Now, consider the same situation but at this time A uses R9 first to call 911 then tries to use Three Way Calling feature in R10 to put 911 on hold and inform his son of the situation. In this case the system will not execute the Three Way Calling as the 911 service prevents anyone from putting it on hold. Consider this as system behaviour B2 when 911 executes first then the Three Way Calling. Obviously B1 ≠ B2 because in B1 both TWC and 911 are executed successfully but in B2 only 911 is executed successfully.
Parameters Effect	If X13 did not contain the Three Way Calling feature then there is no interaction between R9 and R10.

Table 3.9: Description of the interaction scenario SCR15

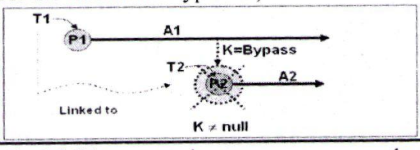
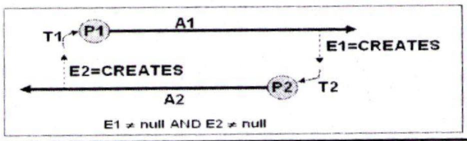
Scenario ID	SCR15
Type of Interaction	TwoInteractingDynamicBehaviourRequirements → Action-PreStateInteractions → Bypass $\|_{C4=LinkedTriggerEvents}$
Detection Guideline	IF {(R1.TriggerEvent ~> R2.TriggerEvent) AND (R1.Action Bypass R2.PreState)} Then {R1 interacts with R2 under the interaction type t15}  T1 P1 A1 K=Bypass T2 R2 A2 Linked to K ≠ null
Example	• R5 (revisited from SCR10): R5:"As a security measure, secure the doors and windows of a house by having them closed starting at time (X5=11:00 pm) for (X6=6 hours)" • R13: When the intruder alarm is triggered and goes on then the security control unit is frozen and it can be unfrozen only by a PIN • Interaction: Suppose that R13 is triggered and starts executing. One part of R13's action is to freeze the security control unit to prevent an intruder from disabling the alarm, which in that case would look like a system glitch, or opening doors and windows to escape. Now this would bypass the prestate of R5 and it will not allow the trigger event of R5 to trigger R5 simply because the whole security control unit including doors and windows is completely frozen (in another abnormal state). Therefore it can be said that the action of R13 bypasses the prestate of R5
Parameters Effect	If X6 in the requirement R5 was set to 0 hours then, the user is disabling the requirement from executing and in this case there is no interaction, as R5 is not supposed to do anything and hence it is not affected by R13.

Table 3.10: Description of the interaction scenario SCR16

Scenario ID	SCR16
Type of Interaction	TwoInteractingDynamicBehaviourRequirements → TriggerEvent-TriggerEventInteractions → InfiniteLooping $\|_{C5=DualLinkedTriggerEvents}$
Detection Guideline	IF {(R1.TriggerEvent <~~> R2.TriggerEvent) AND (R1.Action CREATES R2.TriggerEvent) AND (R2.Action CREATES R1.TriggerEvent)} Then { R1 interacts with R2 under the interaction type t16}  T1 P1 A1 E1=CREATES E2=CREATES P2 T2 A2 E1 ≠ null AND E2 ≠ null
Example	• R4 (revisited from SCR9):"Increase the temperature inside the house to the preset temperature (X4=22) when temperature reading from thermostat is ≤ {(X4=22) – 2} degrees" • R12: "Open the windows in locations (X16=LivingRoom and BedRoom) to decrease the temperature when the thermostat reading is ≥ (X17=22) degrees. Then close them again when the thermostat reading is ≤ {(X17=22) – 2} degrees" • Interaction: Suppose that the house temperature is now at 24 degrees then R12 is triggered and the windows are opened to decrease the temperature inside the house to 20 degrees. Once the temperature is at 20 degrees then the windows closes but also R4 is triggered (i.e., the action of R12 dropped the temperature to 20 which means that it created the trigger event of R4). Now R4 starts executing and pumps hot air to increase the temperature back to 22. Once the temperature reaches 22 then R12 is triggered and starts executing again (i.e., the action of R4 created the trigger of R12 which is to have a temperature ≥ 22 degrees).The preceding process repeats indefinitely. It is noted that the first requirement is created by a person who wants to keep the house temperature at 22 degrees while the second requirement is created by someone who wants to keep the house temperature at 20 degrees. This is understandable in a multi occupant smart home
Parameters Effect	If X17 is changed to 24 degrees then looping chain is broken. Also if X4 is changed to other values then the looping is broken. It must be noted that R4 and R12 are representative of increasing and decreasing temperature requirements. The numbers are just for clarification. The interaction would still occur if X4 was 22.5 for example as the small fractions cannot be precisely achieved when increasing or decreasing the temperature

3.4 Comparison of the Proposed Taxonomy to Already Existing Taxonomies

In this section, we compare the proposed interaction taxonomy to the following already existing interaction taxonomies:

1. Feature interaction benchmark for Intelligent Networks –proposed by Cameron *et al.* in 1994 [20]. Cameron *et al.* presents in [20] two approaches for categorizing interactions which will be denoted by C-1 and C-2, respectively.
2. Interaction taxonomy for services of networked appliances – proposed by Kolberg *et al.* in 2003 [21] and denoted by K.

3. Interaction taxonomy for policies – proposed by Reiff-Marganiec *et al.* in 2004 [68] and denoted by R.

The three taxonomies mentioned above were chosen because they are cited frequently (first taxonomy) or there is close similarity with our proposed interaction taxonomy (second taxonomy) or they are very recent (the third taxonomy). The comparison will be based on:

- The method used for categorizing interactions (e.g., nature of interactions)
- The main focus of the taxonomy (e.g., telecommunication telephony features)
- The number of interaction categories and interaction types proposed in each interaction taxonomy
- The number of examples presented to illustrate each interaction category
- The number of presented examples addressed by our proposed interaction taxonomy and whether there are any examples missed and not addressed by our proposed interaction taxonomy.

Using the criteria mentioned above, the results of the comparison are summarized in Table 3.11. However, for brevity of presentation in the body of the thesis, the details regarding the number of addressed examples by our proposed interaction taxonomy (fifth row of Table 3.11) are presented in Appendix C using Tables C.1, C.2, and C.3

Table 3.11: Comparing the proposed taxonomy to other existing taxonomies

	C		K	R	S
	C-1	C-2			
Method of Categorization	**Nature of interactions**	**Cause of interactions**	**Cause of interactions**	**Nature of interactions**	**Cause of interactions**
Main Focus	Telecommunic-ations Intelligent Networks	Telecommunica-tions Intelligent Networks	Smart homes networked devices	Policies	General (with restriction on implementation interactions)
Number of Interaction Categories	5 main categories	3 Main Categories 12 subcategories	4 main categories	5 main categories 19 subcategories	9 main categories 24 subcategories 37 types
Number of presented examples	22	22 (same ones used in C-1)	5	10	37
Number of examples addressed by proposed taxonomy	Addressed: 18 Missed: 4 (implementation interactions)	Addressed: 18 Missed: 4 (implementation interactions)	Addressed: 5 Missed: 0	Addressed: 10 Missed: 0	N/A
C: Cameron *et al.* taxonomy [20] **C-1:** Cameron *et al.* taxonomy - first approach **C-2:** Cameron *et al.* taxonomy - second approach			**K:** Kolberg *et al.* taxonomy [21] **R:** Reiff-Marganiec *et al.* taxonomy [68] **S:** Shehata *et al.* taxonomy (proposed taxonomy)		

From Table 3.11, the following points are evident:

- The proposed interaction taxonomy categorizes interactions according to the cause of interactions. This satisfies the objective of our proposed taxonomy which is to present where, how, and why interactions occur. This is most beneficial in understanding the technical aspects rather than the social aspects of interactions and also facilitates the definition of detection guidelines for interactions between two requirements.
- The proposed interaction taxonomy starts by categorizing interactions into high-level main interaction categories in a similar way as the other taxonomies do. This helps provide a general understanding of the possible interactions. However, the proposed taxonomy provides more in-depth details regarding the subcategories and types of interactions that are abstract or do not exist in other taxonomies.

- The proposed interaction taxonomy is able to address all examples presented in other taxonomies except for the 4 missed interactions under the Cameron *et al.* taxonomy [20]. Those 4 missed interactions are caused by the way the system was implemented and not by the requirements and therefore are intentionally outside the scope of the proposed interaction taxonomy.

3.5 Limitations of the Proposed Interaction Taxonomy

The proposed interaction taxonomy has the limitation of not being able to address deep design or implementation interactions. The taxonomy is designed to address interactions at the requirements and early design stages of software systems. Hence, all implementation interactions are missed. However, in the majority of cases most of the critical interactions manifest themselves during the requirements engineering phase of the software lifecycle [13] and hence can be captured by the proposed interaction taxonomy. Another limitation is when detecting interactions that involve resources. The definition of resources uses only three attributes: availability, performance and interface. This reduces the number of interaction types to those interactions that involve these attributes. However, resources vary heavily and the number of attributes that can be used to describe them can be very large. Therefore we limited the number of attributes to the common ones which are: Availability, performance, and interface. However, the proposed interaction taxonomy is expandable and can be extended by adding new attributes as needed to be suitable for other domains.

3.6 Summary

This chapter presented a general taxonomy for identifying requirements interactions in software systems. In total, the proposed interaction taxonomy has 9 main interaction categories, 24 interaction subcategories, 37 interaction types, and 37 interaction scenarios that contained 37 interaction detection guidelines that can be used to detect the corresponding interaction types.

The proposed interaction taxonomy is novel in the following sense: It is a general taxonomy that can be applied in any domain rather than being oriented towards a specific domain. This can be seen from the principle of representing the system under consideration using general and domain independent attributes. Hence, it can be considered as the first domain-independent requirements interaction taxonomy. Also, the taxonomy provides 37 interaction scenarios that give a detailed description of when two requirements are considered interacting. The 37 interaction scenarios provide also 37 detection guidelines that can be used to detect the different interaction types.

The proposed interaction taxonomy was compared to other existing taxonomies in the literature and not only was it able to address the interaction issues in those taxonomies but it also contained many other interaction types that have not been captured by other taxonomies.

CHAPTER FOUR: IRIS: IDENTIFYING REQUIREMENTS INTERACTIONS USING SEMI-FORMAL METHODS

4.1 Introduction

This chapter introduces an approach for detecting requirements interactions called Identifying Requirements Interactions using Semi-formal methods (IRIS). As the name already indicates, the approach uses semi-formal methods such as tables, graphs, interaction scenarios, and human judgment to identify interactions between software requirements. In contrast to several other approaches that have been surveyed in Chapter 2, IRIS is a customizable and domain-independent approach. This means that IRIS can be customized to detect interactions in different domains and at different levels of abstraction and thoroughness using semi-formal methods. As a result, IRIS is an approach that fills the gap between existing informal and formal interaction detection approaches.

Section 4.2 gives an overview of some basic concepts of the proposed IRIS approach. Then, Section 4.3 provides a detailed description of IRIS along with a description of the steps that must be performed when applying IRIS. In Section 4.4, a discussion of the advantages of using IRIS for detecting interactions in software systems is provided. Section 4.5 lists the limitations of IRIS. Section 4.6 compares IRIS to other semi-formal approaches found in the literature. Finally, in Section 4.7, the chapter is summarized.

4.2 Overview of IRIS

4.2.1 General Outline

Figure 4.1 shows how IRIS is applied to detect interactions when developing a software system. IRIS is applied during the Requirements Engineering phase. The requirements can be all new, or some new requirements are added to a set of already existing system requirements, or reusable requirements are tailored and added to the system requirements. IRIS is a semi-formal approach which means that it involves graphical and tabular representations and human subjective judgment involving an analyst. The analyst is a regular human developer who must be knowledgeable and experienced in the application of IRIS to ensure the successful application of IRIS otherwise the whole process can fail. However, the analyst does not have to be a domain expert.

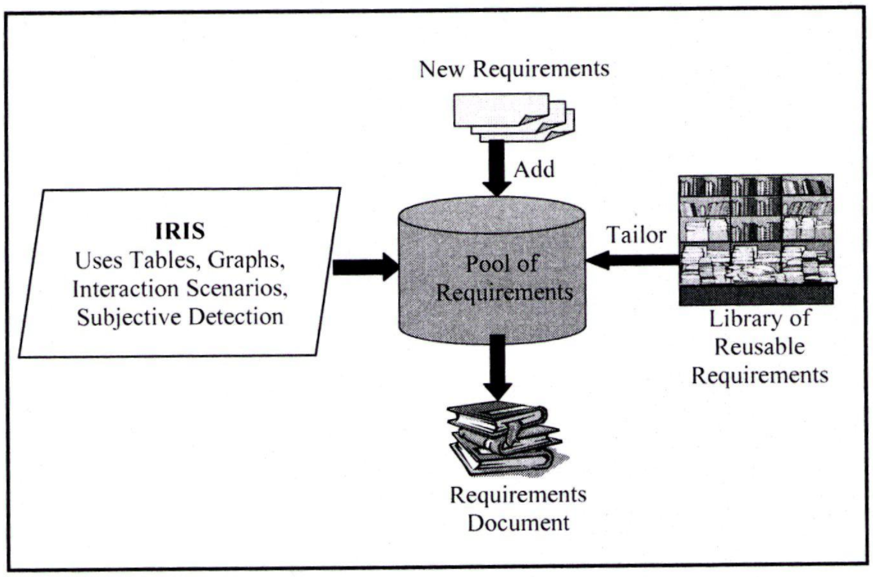

New Requirements
Add
IRIS
Uses Tables, Graphs, Interaction Scenarios, Subjective Detection
Pool of Requirements
Tailor
Library of Reusable Requirements
Requirements Document

Figure 4.1: Application of IRIS to detect interactions when developing a software system

4.2.2 Detecting Interaction with IRIS at Different Abstraction Levels

Originally, IRIS was developed to detect interactions between requirements. However, IRIS has also been successfully applied to detect interactions between features, as well as between policies. This is because IRIS uses the concept of attributes which was discussed in Chapter 3. Usually, a software system being developed is described with a long textual description using requirements, features or policies. Regardless of the way a system is described, the system will still consist of three main components: a static view represented by system axioms, a dynamic view represented by dynamic behaviour, and an environmental view which is represented by resources. Since any of these elements can be represented using attributes as discussed in Chapter 3, this enables IRIS wide applicability for detecting interactions between requirements, between features, or between policies as demonstrated by the case studies presented in Chapters 6, 7, and 8 respectively.

In this chapter, a description of IRIS is given based on the assumption that IRIS is being applied to detect interactions at the requirements level. Hence, the word "Requirements" is being used throughout the remainder of this chapter when explaining and describing IRIS and its steps. The same description is also valid for features and policies.

4.2.3 IRIS Customizability

As mentioned in the introduction to this chapter, IRIS is a customizable approach. The customizability of IRIS means that it can detect interactions in any domain and at different levels of thoroughness. To achieve such a goal, IRIS was designed with a basic core as well as extension hooks that allow expansion through the addition of plug-ins attached to the hooks. The basic core of IRIS consists of several main steps, tables, graphs, and interaction scenarios that always have to be applied regardless of the domain or the type of system under consideration or the abstraction level on which IRIS is being applied (e.g., requirements, features, or policies).

The basic core of IRIS is already capable of detecting critical interactions within a software system, such as non-determinism and conflicting actions being executed simultaneously or sequentially. However, plug-ins can be used to customize IRIS for new domains and also enhance the obtained results by providing more steps, tables, interaction scenarios, etc to detect interactions more thoroughly. This chapter focuses on the basic core of IRIS and its associated steps, tables, graphs, and interaction scenarios. However, a discussion on the customization concept of IRIS and the different plug-ins is provided in Chapter 5.

4.3 IRIS: Class Model and Description

4.3.1 A Class Model for IRIS

The basic core of IRIS consists of several main steps, tables, graphs, and interaction scenarios that are applied regardless of the domain or the type of system under development. IRIS basic core is a systematic approach composed of six ordered steps that 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 interaction scenarios to detect interactions. IRIS basic core is graphically presented in Figure 4.2. Each block represents a class in the figure. Each class has three parts, the top part contains the name of the class, while the middle part contains any requirements attributes that are used within this class, and finally the bottom part contains steps that are executed in this class.

The figure starts by having a requirement document which is represented by the "Req. Document" class. The requirements contained in the requirements document are then classified into system axioms, dynamic behaviour, and resources. This classification is represented by the step Classify_Requirements(Reqs). It is worth mentioning that the output of the classification is zero or more system axioms, zero or more resources, and one or more dynamic behaviour requirements. This is because a system can consist of dynamic behaviour requirements without system axioms or resources as seen from the case study in Chapter 7. But a system cannot consist of only properties without description of the behaviour of the system.

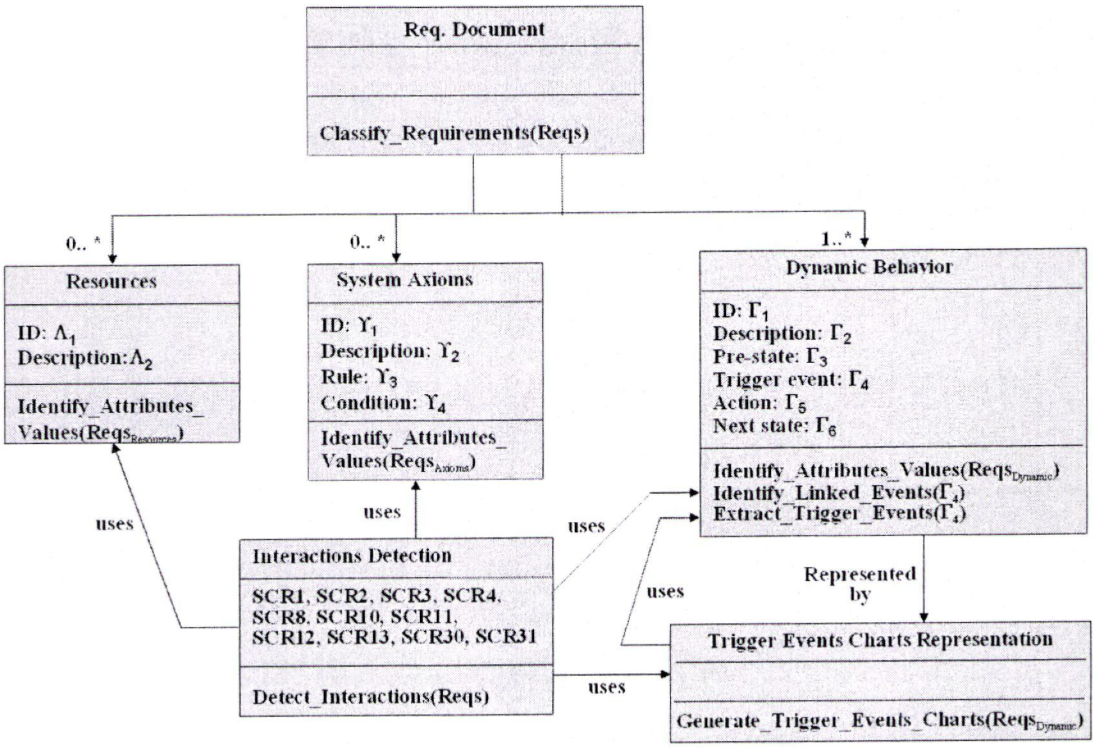

Req. Document
Classify_Requirements(Reqs)
0.. *
0.. *
1..*
Resources
ID: Λ_1
Description: Λ_2
Identify_Attributes_Values(Reqs$_{Resources}$)
System Axioms
ID: Υ_1
Description: Υ_2
Rule: Υ_3
Condition: Υ_4
Identify_Attributes_Values(Reqs$_{Axioms}$)
Dynamic Behavior
ID: Γ_1
Description: Γ_2
Pre-state: Γ_3
Trigger event: Γ_4
Action: Γ_5
Next state: Γ_6
Identify_Attributes_Values(Reqs$_{Dynamic}$)
Identify_Linked_Events(Γ_4)
Extract_Trigger_Events(Γ_4)
uses
uses
uses
uses
uses
Represented by
Interactions Detection
SCR1, SCR2, SCR3, SCR4, SCR8, SCR10, SCR11, SCR12, SCR13, SCR30, SCR31
Detect_Interactions(Reqs)
Trigger Events Charts Representation
Generate_Trigger_Events_Charts(Reqs$_{Dynamic}$)

Figure 4.2: A class model of the basic core of IRIS

The "Resources" class has two attributes Λ_1 and Λ_2 and one step which identifies the values of these attributes for all resources. The "System Axioms" class uses attributes Υ_1-Υ_4 and has one step which identifies the values for these attributes for all system axioms. The "Dynamic Behaviour" class uses attributes Γ_1-Γ_6 and has three steps. The step Identify_Attributes_Values(Reqs$_{dynamic}$) identifies the attributes values for all the dynamic behaviour requirements. The step Extract_Trigger_events(Γ_4) extracts all unique trigger events from the different values of the attribute "Trigger event" and lists them in a separate table. The step Identify_Linked_Events(Γ_4) identifies all linked events and lists them in a separate table. The class "Trigger Events Charts Representation" has only one step, Generate_Trigger_Events_Charts(Reqs$_{Dynamic}$), which creates trigger events charts for all dynamic behaviour requirements. Finally, the "Interaction Detection" class detects interactions between requirements through the step Detect_Interactions(Reqs). In this class, the step Detect_Interactions(Reqs) uses 11 of the 37 interaction scenarios, namely: SCR1, SCR2, SCR3, SCR4, SCR8, SCR10, SCR11, SCR12, SCR13, SCR30, and SCR31 to detect interactions between requirements. The details of these interaction scenarios are provided in Chapter 3 and Appendix B.

Figure 4.2 shows that the basic core of IRIS contains six steps. These steps are ordered in such a manner that the translation of requirements into graphical and tabular representations is gradually achieved. The objective of these representations is to facilitate the application of the interaction scenarios in the sixth step of IRIS.

4.3.2 Step 1: Requirements Classification

As stated in Chapter 3, any system will be decomposed into a static view represented by system axioms, a dynamic view represented dynamic behaviours, and an environmental view represented by resources. The first step is used to classify requirements contained in the requirements document into one of three categories:

1. System axioms
2. Dynamic behaviours
3. Resources

Step 1 is performed by having the analyst examine the textual description of the requirements of the system and determine if a requirement is a system axiom or a dynamic behaviour or a resource. If the requirement describes a certain property of the system that has to be preserved, then the requirement is a system axiom. If the requirement describes how the system should respond in terms of state changes and actions needed to be taken when a specific trigger occurs, then this requirement is a dynamic behaviour requirement. Finally, if a requirement describes system specific resource requirements, then this is a resource. It must be noted that a requirement can belong to only one category, i.e., a requirement can be either a system axiom or a dynamic behaviour or a resource. As an example, Table 4.1 shows three requirements and the class they belong to.

The requirement classification step is shown in Figure 4.2 by the Classify_Requirements(Reqs) in the Req. Document Class.

Table 4.1: Examples on classifying requirements

Requirement	Requirement Classification
Occupants can control all A/V devices through remote controls	System axiom requirement
Automatically turn on the lights according to a daylight sensor when the night begins.	Dynamic behaviour requirement
The database server shall be available for processing requests more than 99.9% of the time during each week	Resource requirement

4.3.3 Step 2: Requirements Attributes Identification

As mentioned in Chapter 3, the use of attributes was proposed as a general representation to describe system axioms or dynamic behaviour requirements or resources. This step identifies the values of the different attributes of each requirement (either a system axiom or a dynamic behaviour or a resource). For a system axiom there are four basic attributes which are: ID, Description, Rule, and Condition, Υ_1-Υ_4. The two attributes Parameters Υ_5 and Parameters Range Υ_6 are optional attributes and hence are plug-ins that are not part of the basic core of IRIS. In case of a dynamic behaviour requirement, there are 6 basic core attributes which are: ID, Description, Pre-State, Trigger Event, Action, and Next State Γ_1-Γ_6. The two attributes Parameters Γ_7 and Parameters Range Γ_8 are optional attributes and hence are plug-ins that are not part of the basic core of IRIS. Finally, in case of a resource, there are only two attributes that are considered as part of the basic core of IRIS which are: ID Λ_1 and Description Λ_2. The three attributes: Availability, Performance, and Interface, Λ_3-Λ_5, are considered to be plug-ins to IRIS.

Step 2 is performed by the analyst who identifies the different values of attributes for the requirements in each of the three categories determined in step 1.

Step 2 has three tables as output. The first table is called the "System Axioms Attributes Identification" and contains all the system axioms along with the values of the attributes for each system axiom (see Table 4.2 for an example). The second table is called the "Dynamic Behaviour Attributes Identification" and contains all the dynamic behaviour requirements along with the values of the attributes for each dynamic behaviour requirement (see Table 4.3 for an example). The third table is called "Resources Attributes Identification" and contains all resources requirements along with the values of the attributes for each resource requirement (see Table 4.4 for an example).

The requirements attributes identification step is shown by Identify_Attributes_Values(Reqs) which exists in the classes Dynamic Behaviour, System Axioms, and Resources in Figure 4.2.

Table 4.2: System Axioms Attributes Identification

ID	Description	Rule	Condition
R1	Occupants can control all A/V devices through remote controls	Control all A/V devices through remote controls	True
...	...	...	...
...	...	...	...

Table 4.3: Dynamic Behaviour Attributes Identification

ID	Description	Pre-State	Trigger Event	Action	Next State
R2	Automatically turn on the lights according to a daylight sensor when the night begins.	Daylight=True Lights=Off	Night begins	Automatically turn on the lights	Daylight=False Lights=On
…	…	…	…	…	…
…	…	…	…	…	…

Table 4.4: Resources Attributes Identification

ID	Description
R3	The database server shall be available for processing requests more than 99.9% during each week
…	…
…	…

4.3.4 Step 3: Trigger Events Extraction

Step 3 is aimed at identifying and extracting all the different and unique trigger events that can cause dynamic behaviour requirements to execute. This step is performed by looking at the table "Dynamic Behaviour Attributes Identification" that was created in step 2 and by determining all the different unique trigger events from the attribute column "Trigger Event". After that, these trigger events are listed in a separate table called "Trigger Events Extraction" table as shown in Table 4.5. Each trigger event is listed in the table and is given a unique ID (e.g. E1, E2…) with an informal description of the trigger event, as specified in the Trigger Event attribute column of Table 4.3, and a list of which requirements this event is triggering. This table is important as it will be used when creating the trigger events charts in step 5.

Step 3 is shown by Extract_Trigger_Events(Γ_4) in the dynamic behaviour class in Figure 4.2.

Table 4.5: Trigger Events Extraction

Event ID	Event Description	Requirements Triggered by this Event
E1	Night begins	R2
...	...	...
...	...	...

4.3.5 Step 4: Linked Events Identification

To understand step 4 correctly, the concept of linked events must first be introduced. Linked events are trigger events that are connected to each other. Linked events can best be described using an example. For instance, consider the trigger event E_1 stating that "A window is opened". The occurrence of this event will likely cause a change in the temperature of the house and therefore the trigger event E_2, which describes that the temperature of the home has changed, will also be triggered. This means that whenever event E_1 occurs then event E_2 will also occur as a logical consequence after a short time period. This means that event E_1 leads to event E_2 (or event E_2 is linked to E_1) and this is expressed as E_1 ~> E_2 where the curly arrow (~>) indicates that the occurrence of the first event will most likely lead to the occurrence of the second event.

The degree of confidence that the occurrence of the first trigger event will lead to the occurrence of the second trigger event is not of major concern because linked trigger events are used to detect sequential interactions and hence if there is any chance that two events are linked then it is better to mark them as linked trigger events in order not to miss any sequential interactions.

An informal definition of linked events can be given as follows:

"Granted that events can be initiated by a user or a system, event E_2 is said to be linked to event E_1 ($E_1 \rightsquigarrow E_2$) if the occurrence of event E_1 is followed by the occurrence of event E_2 as a logical sequence".

To give a more rigorous definition of linked events, consider the following symbols:

E: Event

U: User

S: System

@E: At the occurrence of the event E

createEvent( ): Function that can represent a user or the system creating an event E

Using these symbols, linked events can be formally defined as follows:

$$(E_1 \sim> E_2) \leftrightarrow (@E_1 \rightarrow (E_2 = S.\text{createEvent}(\,) \vee E_2 = U.\text{createEvent}(\,))) \qquad (4.1)$$

This latter formula can be read as follows "(event E_2 is linked to event E_1) is equivalent to (at the occurrence of event E_1 this will lead to (event E_2 is created by the system S OR event E_2 is created by a user U))".

The linked events definition can also be extended to include transitive linked events. Two events are said to have a transitive link relation when these two events are not linked directly to each other but through one or more linked events. A transitive link between the two events E_1 and E_3 exists as follows:

"If event E_2 is linked to event E_1 AND event E_3 is linked to event E_2 then this leads to event E_3 is said to be linked to event E_1"

This definition can be translated into the following equation using the same notation explained earlier as follows:

$$(E_1 \rightsquigarrow E_2) \wedge (E_2 \rightsquigarrow E_3) \rightarrow (E_1 \rightsquigarrow E_3) \tag{4.2}$$

The purpose of investigating linked events is to detect interactions between requirements that are sequentially related through linked events. This can be seen from the interaction taxonomy introduced in Chapter 3 where there are interaction scenarios for identifying interactions between sequentially executed requirements. Linked events are the mechanism that IRIS uses for identifying requirements that are related sequentially and hence allow the detection of sequential interactions. Moreover, the concept of transitive linked events provides deep and sufficient means for IRIS to identify sequentially related requirements even if these requirements are not related through a direct sequence of events.

Linked events are identified in the 4th step of IRIS. During the 4th step of IRIS, an analyst looks at the table "Trigger Events Extraction" (Table 4.5) that was created in step 3 of IRIS. The trigger events listed in that table are examined in order to identify if the occurrence of an event can lead to the occurrence of another event. In that case the two events are said to be linked events and are they are listed in the "Linked Events Identification" table shown in Table 4.6

Table 4.6: Linked Events Identification

Event ID	Event Description	Linked to	Mathematical Representation
E1	A window is opened	E2	E1~>E2
...	...	...	...

The linked events identification step of IRIS is shown in Figure 4.2 by Idenitfy_Linked_Events(Γ_4) in the dynamic behaviour class.

4.3.6 Step 5: Trigger Events Charts Representation

In step 5, a graphical notation is used to link each trigger event with the dynamic behaviour requirements it triggers. This graphical notation is called "Trigger Events Charts" because it graphically groups together dynamic behaviour requirements that are triggered by the same trigger event. Trigger events charts are very useful as they facilitate the detection of interactions between the requirements, performed in the sixth step of the proposed IRIS approach.

Trigger events charts are created only for dynamic behaviour requirements and are created by having the analyst look at the "Trigger Events Extraction" table which was created in step 3 of IRIS (Table 4.5) to identify which requirements are triggered by the same trigger event. These requirements are then graphically represented as shown in Figure 4.3.

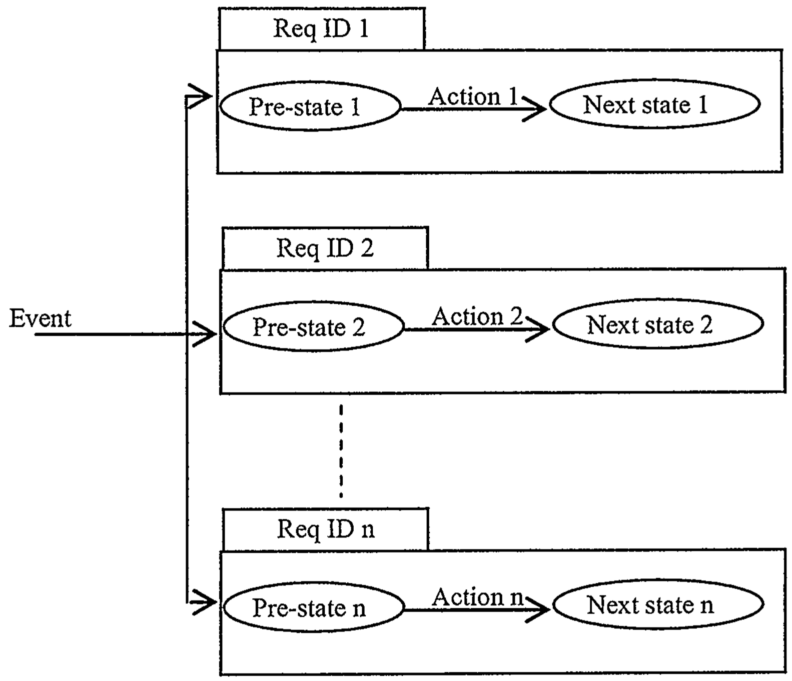


Figure 4.3: Trigger Events Charts

In Figure 4.3, the word Event represents any trigger event that has been identified in step 3. Each requirement that is triggered by this event is graphically represented as a rectangle that shows the following attributes: Requirement ID, Pre-State, Action, and Next State. The values of these attributes for each of the requirements are extracted from the table "Dynamic Behaviour Attributes Identification" (Table 4.3).

The trigger events chart provides a graphical view for the analyst to easily compare and apply the interaction scenarios in the sixth step of IRIS (Interaction Detection) in order to find interactions between requirements that are triggered by the same trigger event or requirements that are triggered by linked events.

The trigger events charts representation step is shown by Generate_Trigger_Events_Charts(Reqs) in the class Trigger Events Charts Representation in Figure 4.2.

4.3.7 Step 6: Interactions Detection

4.3.7.1 General Description

The interaction detection step is the last and final step of IRIS. The interaction detection in this step is subjective which means that the analyst detects interactions between requirements using the different tables and graphs that have been created in steps 2, 3, 4, and 5 and also uses the different interaction scenarios that are part of the general requirements interaction taxonomy described in Chapter 3.

The subjectivity of interaction detection is minimized through the application of the interaction scenarios that help correctly detect interactions between requirements and serve as an experience base for the human analyst. Also, the developed tables and graphs from the previous steps serve as a clear presentation of the information collected so far during the detection step. Therefore, with these interaction scenarios being applied on the developed tables and graphs, the subjectivity of the approach is reduced. According to the interaction taxonomy in Chapter 3, there are 9 main interaction categories:

1. Interactions between two system axioms
2. Interactions between a system axiom and a dynamic behaviour requirement
3. Interactions between two dynamic behaviour requirements
4. Interactions between a system axiom and a resource
5. Interactions between a dynamic behaviour requirement and a resource
6. Interactions between two resources
7. Interactions between a dynamic behaviour requirement and a system axiom
8. Interactions between a resource and a system axiom
9. Interactions between a resource and a dynamic behaviour requirement

The analyst now tries to find interactions between requirements that fall in these 9 main interaction categories by applying the interaction scenarios provided under these categories to detect interactions. However, not all interaction scenarios are always applicable because some interaction scenarios are plug-ins and are not part of the basic core of IRIS. For example, the interaction scenario SCR16 "Infinite Looping" is aimed at finding interactions between two dynamic behaviour requirements due to infinite looping but at the same time this interaction scenario actually detects interactions due to high level system design problems, and therefore is not always applied (note that the application of interaction scenarios requires time and effort and there might be situations where such a thorough detection is not required). Also, there are interaction scenarios that are applied only in specific cases. For example, there are interaction scenarios aimed at detecting interactions when there are specific requirements for resources availability. These interaction scenarios are applied when the attribute plug-in "Availability" is used (Chapter 5 provides more details on using plug-ins with IRIS).

Based on this discussion, the following interaction scenarios are identified as part of the basic core of IRIS: SCR1, SCR2, SCR3, SCR4, SCR8, SCR10, SCR11, SCR12, SCR13, SCR30, and SCR31 (the complete details of these interaction scenarios are provided in Chapter 3 and Appendix B). These interaction scenarios were chosen because they provide detection of the most common critical interactions at the requirements level based on the different case studies that have been conducted in this thesis or based on the extensive literature survey of current approaches (previously presented in Chapter 2) that was conducted during this research. However, more interaction scenarios can be plugged into IRIS as needed as explained Chapter 5.

The following provides a description of how the 6th step of IRIS is applied using the developed graphs and tables from the previous steps of IRIS and interaction scenarios that are part of the basic core of IRIS.

4.3.7.2 Detecting Interactions According to Main interaction Category ①

The main interaction category number ① provides interaction scenarios for detecting interactions between two system axioms. There are two basic core interaction scenarios, SCR1 and SCR2, that are part of this main interaction category. According to these two interaction scenarios, the analyst is required to examine all the system axioms listed in the "System Axioms Attributes Identification" table.

In order to detect interactions, the analyst compares pair-wise all the system axioms with specific focus on the values of the rule attribute of the two system axioms being compared, to find interactions. According to the first interaction scenario, SCR1, an interaction is detected if the rule attribute of the first requirement overrides the rule attribute of the second requirement. Whereas the second interaction scenario, SCR2, states that an interaction exists between two requirements if the rule attribute of the first requirement has a negative impact on the rule attribute of the second requirement. Whenever the analyst encounters one of these two situations when examining the rule attributes of a pair of system axiom requirements, then these two requirements interact.

4.3.7.3 Detecting Interactions According to Main Interaction Categories ② and ⑦

The main interaction category number ② provides interaction scenarios for detecting interactions that occur between a system axiom and a dynamic behaviour requirement. On the other hand, the main interaction category number ⑦ provides interaction scenarios for detecting interactions that occur between a dynamic behaviour requirement and a system axiom. The main interaction categories number ② and ⑦ were joined together to avoid making the analyst comparing the same two requirements twice first under main interaction category ② and then under main interaction category ⑦.

There are four basic core interaction scenarios to be used which are SCR3, SCR4, SCR30, and SCR31. According to these four interaction scenarios, the analyst is required to examine the table "System Axioms Attributes Identification" and the table "Dynamic behaviour Attributes Identification" developed in step 2. The analyst has to compare pair-wise every system axiom and every dynamic behaviour requirement with the objective of finding interactions based on the four interaction scenarios SCR3, SCR4, SCR30, and SCR31.

4.3.7.4 Detecting Interactions According to Main Interaction Category ③

This main interaction category contains scenarios for detecting interactions between two dynamic behaviour requirements. There are 5 basic core interaction scenarios under this category which are: SCR8, SCR10, SCR11, SCR12, and SCR13. The first three interaction scenarios are used to detect interactions between two dynamic behaviour requirements that are triggered by the same trigger event while the last two scenarios are used to detect interactions between requirements triggered by linked events. The analyst first looks at the trigger events charts developed in step 5 and the linked events identification table developed in step 4 and extracts all unique pairs of requirements that are triggered by the same trigger event or triggered by linked trigger events. These pairs are the ones to be examined for interactions using the five interaction scenarios under this category. This way, the analyst discards unrelated comparisons that will not lead to interaction situations.

The analyst now examines each one of the identified pairs of requirements using the trigger events charts developed in step 5 with the aim of finding interactions between the two requirements in the pair under investigation. The examination is done by applying the five interaction scenarios SCR8, SCR10, SCR11, SCR12, and SCR13 on the two requirements being investigated to see if any interaction can occur between them. For example, according to SCR8, the developer examines the trigger events chart for the two requirements R1 and R2 that are triggered by the same trigger event and see if R1 and R2 have the same pre-sate and have different next states. Whenever such a situation occurs, then these two requirements interact according to SCR8 because this would cause a non-determinism situation in the system.

As another example for detecting interaction between two requirements triggered by linked trigger events, consider E1~>E2 and E1 triggers the requirement R3 while E2 triggers the requirement R4. According to SCR12, the analyst examines R3 and R4 to determine if the action of R3 overrides the action of R4 or vice versa. If such a situation occurs, then the two requirements R3 and R4 interact according to the interaction scenario SCR12.

4.4 Advantages of the Proposed IRIS Approach

In Section 4.3, IRIS has been proposed as a semi-formal approach for detecting requirements interactions. This section focuses on highlighting the main advantages and characteristics of IRIS:

- IRIS is a semi-formal approach for detecting interactions. This means that it does not require any heavy mathematical modeling of the system under investigation as opposed to formal methods.

- IRIS reduces the number of necessary pair-wise comparisons that have to be performed between all requirements in textual form. This is very important and crucial as "The analysis of feature interactions is almost impossible in complex system because the number of combinations to be analyzed grows exponentially with the number of features" [140]. IRIS discards irrelevant comparisons between requirements that will not lead to interactions (an irrelevant comparison is a comparison that contains two requirements that are not triggered by the same trigger event or by linked events). This can result in a clear reduction in the number of comparisons as demonstrated in the case studies (chapters 6, 7, and 8). Although this reduction in number of comparisons cannot be translated directly into equivalent reduction in cost and time due to the fact that there will always be an overhead due to the application of IRIS, but IRIS, as a structured approach, is likely to increase the number of detected interactions. The increased number of detected interactions will compensate for the additional time and effort of applying IRIS. Also, the reduction in number of comparisons favours the proposed IRIS approach.
- IRIS is not limited to a specific domain (e.g. the telecommunications domain) but is domain independent. This is obvious through:
  - The general representation notations adopted in the different steps of IRIS.
  - The general interaction taxonomy that provides general interaction scenarios applied in the sixth step of IRIS.
  - The different case studies from different domains in which IRIS has been applied to detect interactions.

- IRIS adopts the terminology introduced by Robinson *et al.* [11] which extends the definition of feature interaction to Requirements Interaction Management (RIM). This means that IRIS focuses on detecting interactions between requirements during the requirements engineering stage to save costly repairs at later stages.
- IRIS is capable of detecting interactions at different abstraction levels. In this thesis, IRIS was able to detect interactions at the requirements level (case study in Chapter 6), at the features level (case study in Chapter 7), and at the policies level (case study in Chapter 8).
- IRIS is a customizable approach that can be extended by adding plug-ins to enhance its performance and detection accuracy. More discussion on this point is provided in the next chapter.
- The tables created in IRIS allow a comprehensive representation and visualization of the requirements of the system in a structured format. The creation of these tables requires a good understanding of the requirements forcing the developer to clearly think about requirements which will likely improve them. This is because when a developer cannot easily identify the values for the attributes of a requirement, then this means that the requirement under investigation is incomplete or ambiguous. Hence, the developer has to go back to the stakeholder of this requirement to enhance and improve it.
- Using trigger events charts makes detection of interactions between dynamic behaviour requirements easier. For instance, requirements that are triggered by the same event are grouped together in the trigger events charts. Therefore detecting

interactions can be easily done by examining the different actions and states of the requirements according to the interaction scenarios being used.

4.5 Limitations of the Proposed IRIS Approach

After discussing the benefits and advantages of using IRIS, it is also important to discuss its limitations. The discussion of these limitations is as important as the approach itself so that IRIS will not be used beyond its capabilities resulting in unsatisfactory performance. The following summarizes the limitations of IRIS:

- IRIS is an offline detection approach which means that it cannot be used to detect interactions in an interactive runtime environment. However, IRIS can be used to detect interactions offline and then implement the obtained results in any online detection approach as a knowledge base. Moreover, this limitation can be compensated for by implementing the general interaction scenarios, which are part of IRIS, in any online interaction detection approach.
- IRIS is a detection approach only which means that IRIS does not provide suggestions for the resolution of the detected interactions. The resolution has been intentionally left out of IRIS because different resolutions are available based on the different stakeholders involved. Any suggested resolutions for the detected interactions must involve an iterative negotiation process between the stakeholders involved in setting the interacting requirements.
- IRIS is a semi-formal approach that has subjectivity in the interaction detection step. This means that it is not guaranteed to detect all the interactions in a system. Therefore, IRIS is only recommended for non-critical systems such as

commercial PC software, telecommunications features, and smart homes. However, IRIS can be used as a first stage application to filter as many interactions as possible at the early stage of requirements engineering. Then, once the necessary design and implementation details are available, formal approaches, such as SDL, can be applied to the system to have a more through interaction detection. This way, IRIS can help detect interactions early and avoid high repair cost due to late detection in the software life cycle.

- IRIS is not suitable for detecting detailed design and implementation interactions. This is obvious as IRIS was designed originally to detect interactions between requirements during the requirements engineering phase. This limitation was slightly compensated for by the ability of IRIS to detect high level design interactions such as infinite looping. Still, detailed design and implementation interactions are beyond the capabilities of IRIS. It is worth mentioning that IRIS can still be used as a front end filtering approach to detect interactions as early as possible.
- IRIS can detect only interactions between two requirements (2-way interactions) but it cannot detect interactions that are cause by 3 requirements together (3-way interactions). It is worth mentioning that 3-way interactions are rare and have not been thoroughly addressed in the literature. To the authors' best knowledge, 3-way interactions have been addressed only in the work by Hall [141], Samborski [142], and Kawauchi *et al.* [104].

4.6 Comparing IRIS to other Semi-Formal Approaches in the Literature

To conclude presenting the proposed IRIS approach, a comparison is made between IRIS and other semi-formal approaches that were identified in the literature. There are 7 semi-formal approaches that have been compared to IRIS as shown in Table 4.7. The full details of these approaches that are being compared to IRIS can be found in Chapter 2. However, Table 4.7 compares all the semi-formal approaches including IRIS to highlight the advantages and limitations IRIS over the other approaches. It must be noted that the advantages from the comparison in Table 4.7 are not the only advantages but are added to the list of advantages listed in Section 4.4

Table 4.7: Comparing IRIS with other Semi-Formal Approaches

Criteria	IRIS	Wakahara *et al.* [38]	Mierop *et al.* [39]	Kimbler *et al.* [40]	Dankel *et al.* [41]
Notation Used	Tables, Trigger events charts	MSC	OO	-	High level predicates
Approach Type	Offline	Offline	Offline	Offline	Offline
Application Domain	General	Telecomm	Telecomm	Telecomm	Telecomm
Address System Properties Interactions	Yes	No	No	No	Yes
Address Resources Interactions	Yes	No	No	Yes	Yes
Human Involvement	Regular human developer	Expert with knowledge of telecomm. and MSC	Expert with knowledge of telecomm. and OO.	Expert with knowledge of telecomm.	Designers and Experts
Application Phase	Req. and high level Design	Design	Design	Req. and Design	Req. and Design
Experience Factor	General interaction scenarios	Knowledge bases with data on telecomm.	Human expertise	Human expertise	Human expertise
Number of case studies reported in the literature	3	0	0	0	0
Other Specific Limitations	Do not address deep design and implementation related interactions, Not recommended for critical systems	Knowledge used in the DB are very abstract, Integration of developed MSC is very hard, Not recommended for critical systems	Detects limited types of interactions, representation of telecomm in OO is very hard, Not recommended for critical systems	uses serious simplifications in the ESTI/NA6 specification with no proof of validity. Not recommended for critical systems	Based on natural language processing, Behavioural interactions are detected informally, Not recommended for critical systems

Table 4.7-Continued: Comparing IRIS with other semi-formal approaches

Criteria	IRIS	Kuisch *et al.* [42]	Keck [43]	Kimbler and Sobrik [44]
Notation Used	Tables, Trigger events charts	BCSM	BCSM	Use Case Models
Approach Type	Offline	Offline	Offline	Offline
Application Domain	General	Telecomm	Telecomm	Telecomm
Address System Properties Interactions	Yes	No	No	No
Address Resources Interactions	Yes	Partially	Yes	Yes
Human Involvement	Regular human developer	Expert with knowledge of telecomm. and BCSM	Human developer	Expert with knowledge of telecomm. and Use Case Models
Application Phase	Req. and high level Design	Design	Design	Req. and Design
Experience Factor	General interaction scenarios	Human expertise	Criteria with Rules about telecommunications scenario prone interactions	Human expertise
Number of case studies reported in the literature	3	0	1	0
Other Specific Limitations	Don not address deep design and implementation related interactions, Not recommended for critical systems	Specification in BCSM is not an easy task, The reference does not describe types of resource related interactions that can be detected, Not recommended for critical systems	The generated list contains only interaction prone scenarios and this list must be analyzed by another detection approach for deciding which features are really interacting, The criteria used for identifying interaction prone scenarios is limited, Specification in BCSM is not an easy task, Not recommended for critical systems	The created use case models cannot cover all possible usage scenarios, The final interaction detection relies totally on experience with limited definition of when two features interact, The criteria used for identifying interactions between features is limited, Not recommended for critical systems

4.7 Summary

This chapter presented the proposed semi-formal approach IRIS for detecting requirements interactions. IRIS uses tables and graphs along with interaction scenarios to detect interactions. IRIS is a systematic six step approach that can detect interactions in any domain. IRIS is also a customizable approach which means that it can have plug-ins attached to its basic core to extend and enhance its capability and increase its interaction detection thoroughness. Chapter 5 provides more discussion on the concept of customization for IRIS along with details of what and how the different plug-ins can be hooked to the basic core of IRIS to extend it and enhance its capability.

CHAPTER FIVE: IRIS CUSTOMIZATION

5.1 Introduction

This chapter continues on the previous chapter with a focus on IRIS customization. This chapter describes how IRIS can be customized, the different plug-ins are that can be used with IRIS, and how they can be used and inserted to extend the basic core of IRIS.

IRIS plug-ins are considered to be a very powerful feature in IRIS that can be used to extend the performance of IRIS, increase the scope and thoroughness of interaction detection to include design and resource interactions, make IRIS applicable to new domains, and cope with any specific future needs by system developers.

This chapter is structured as follows: Section 5.2 describes the concept of customizing IRIS and presents its advantages. In Section 5.3, IRIS hooks are described as insertion points for the different plug-ins. It also contains a description of the characteristics of the hooks used in IRIS. Section 5.4 gives details regarding the different plug-ins that can be attached to the hooks. This includes a description of the general structure of the plug-ins, and how plug-ins can be inserted to specific hooks and be integrated as part of the whole approach. Finally, Section 5.5 summarizes this chapter.

5.2 The Concept of IRIS Customization

IRIS was designed to be a domain independent approach that can detect interactions at different levels of thoroughness between software requirements using semi-formal methods. The challenge was to achieve this objective without creating a complicated approach. For this reason, IRIS consists of a basic core that can be applied regardless of the domain and is sufficient by itself to detect critical interactions within a software system. This main core can then be supplemented with different plug-ins to extend it and enhance its capabilities and ensure its successful application in new domains where special needs may arise.

The advantages of extending IRIS with plug-ins can be summarized as follows:

- IRIS basic core is a simple approach that can be easily applied in any domain to detect critical interactions. Hence, the analysts can easily learn how to use and apply IRIS.
- The analyst only has to perform steps and create those tables and graphs that necessary to detect interactions that meets his needs. For example, if the analyst does not want to detect resources interactions, then s/he does not have to create and fill the resources attributes identification tables nor apply interaction scenarios related for the detection of resources interactions. This will greatly reduce the overhead of applying IRIS.
- The created plug-ins can provide different levels of thoroughness for detecting interactions. It is up to the analyst to decide the required level for detecting interactions. If the system is to be thoroughly analyzed, more plug-ins are to be used.

- Some of the created plug-ins are used to add the optional attributes that were described in Chapter 3 (e.g., Parameters and Parameters Range) to system axioms and dynamic behaviour requirements.
- So far 10 plug-ins have been created based on the needs identified from the case studies conducted in this research. However, additional plug-ins can be created by analysts to accommodate new needs when IRIS is applied in new domains. This is a very powerful feature, as IRIS is no longer a static approach that might get useless over time, but can evolve over time. The analysts only have to watch that they follow the general structure and format of plug-ins to ensure the integrity and successful application of IRIS.

5.3 IRIS Hooks

5.3.1 Overview

IRIS was built with a basic core that consists of six main steps. Using these six steps, the requirements are gradually translated into a graphical and tabular representation and finally specific interaction detection scenarios are applied to detect interactions.

In addition to these six steps, tables, graphs, and interaction scenarios, the basic core of IRIS also contains specific point, *so-called Hooks*, into which plug-ins can be hooked to extend IRIS. Figure 5.1 presents the basic core of IRIS and the different hooks.

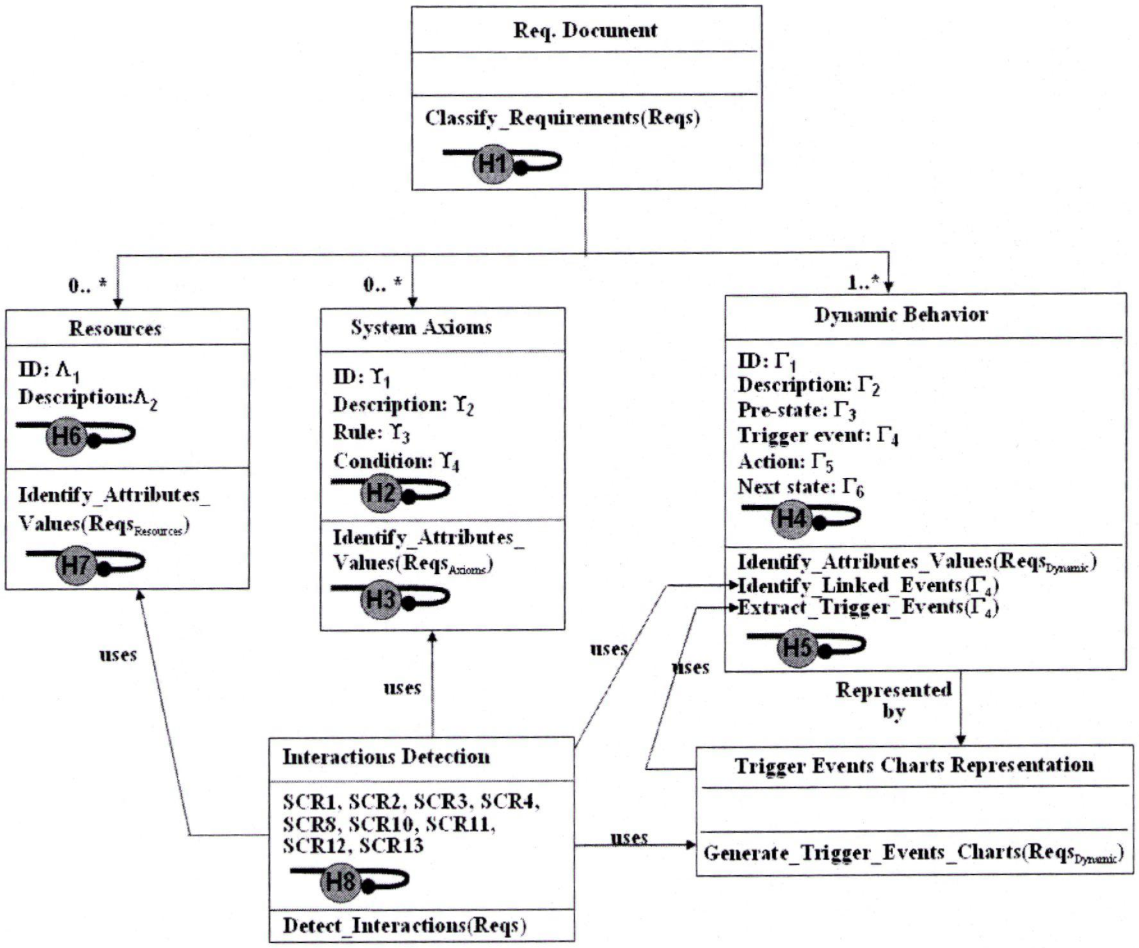

Req. Document
Classify_Requirements(Reqs)
H1
0.. *
0.. *
1..*
Resources
ID: Λ1
Description:Λ2
H6
Identify_Attributes_Values(ReqsResources)
H7
System Axioms
ID: ϒ1
Description: ϒ2
Rule: ϒ3
Condition: ϒ4
H2
Identify_Attributes_Values(ReqsAxioms)
H3
Dynamic Behavior
ID: Γ1
Description: Γ2
Pre-state: Γ3
Trigger event: Γ4
Action: Γ5
Next state: Γ6
H4
Identify_Attributes_Values(ReqsDynamic)
Identify_Linked_Events(Γ4)
Extract_Trigger_Events(Γ4)
H5
uses
uses
uses
uses
uses
Represented by
Interactions Detection
SCR1, SCR2, SCR3, SCR4, SCR8, SCR10, SCR11, SCR12, SCR13
H8
Detect_Interactions(Reqs)
Trigger Events Charts Representation
Generate_Trigger_Events_Charts(ReqsDynamic)

Figure 5.1: Basic core of IRIS showing points of the different hooks

5.3.2 Hooks Characteristics

The hooks which are represented by H1, H2, H3, H4, H5, H6, H7, and H8 in Figure 5.1 are insertion points for plug-ins. Each hook has a unique name that starts with an H followed by a unique number to identify this specific hook. The numbering order used is arbitrary and is of no importance. The locations of the hooks were chosen based on:

- The need to add more attributes such as the optional attributes described in Chapter 3 to describe system elements (e.g., system axioms, dynamic behaviour requirements, and resources). This can be achieved through the hooks H2, H4, and H6.
- The need to extend IRIS to ensure its successful application in a diverse range of domains. This need was obvious from the case studies conducted throughout this research. This can be achieved through the hooks H1 and H3.
- The need to detect more thoroughly interactions using more interaction scenarios other than the basic core interaction scenarios. This can be achieved through the hook H8.
- The need to ensure the ability of IRIS to be extended to accommodate any potential future needs. This can be achieved through the hooks H5 and H7.

In the following we give details on the characteristics of each hook and what plug-ins can be inserted through each of these hooks.

5.3.2.1 Characteristics of Hook H1

Hook H1 is an insertion point to add steps that need to be performed before the application of the basic core steps of IRIS. For this reason H1 is located in the "Req. Document" at the top of the IRIS class model as seen in Figure 5.1.

Hook H1 will accept only the insertion of plug-ins of type STEP (section 5.4.2) and integrates them with the basic core steps of IRIS. The order of execution of the new inserted plug-ins through hook H1 is prior to the execution of IRIS step 1.

5.3.2.2 Characteristics of Hook H2

Hook H2 is an insertion point to add attributes that are needed to fully represent system axioms in the case that a system contains more data that cannot be represented by the basic system axioms attributes. Examples of such attributes are the optional attributes "Parameters" and "Parameters Range" discussed in Chapter 3. For this reason H2 is located in the class "System Axiom" with other basic system axioms attributes as seen in Figure 5.1.

Hook H2 accepts only the insertion of plug-ins of type ATTR (ATTR stands for attributes) and integrates them with other basic system axioms attributes.

5.3.2.3 Characteristics of Hook H3

Hook H3 is an insertion point that allows the addition of steps that might need to be performed on system axioms. For this reason H3 is located in the class "System Axioms" with other basic system axioms steps as seen in Figure 5.1.

The Hook H3 accepts only the insertion of plug-ins of type STEP and adds them to other basic system axioms steps.

5.3.2.4 Characteristics of Hook H4

Hook H4 is an insertion point to add attributes that are needed to fully represent dynamic behaviour requirements in the case that a system contains more data that cannot be represented by the basic dynamic behaviour attributes. Examples of such attributes are the optional attributes "Parameters" and "Parameters Range" discussed in Chapter 3. For this reason H4 is located in the class "Dynamic behaviour" with other basic dynamic behaviour attributes as seen in Figure 5.1.

The Hook H4 only accepts the insertion of plug-ins of type ATTR and integrates them with other basic dynamic behaviour attributes.

5.3.2.5 Characteristics of Hook H5

Hook H5 is an insertion point to add steps that might be needed to be performed on the dynamic behaviour requirements. For this reason H5 is located in the class "Dynamic Behaviour" with other dynamic behaviour steps as seen in Figure 5.1.

The Hook H5 accepts only the insertion of plug-ins of type STEP and adds them with other dynamic behaviour steps.

5.3.2.6 Characteristics of Hook H6

Hook H6 is an insertion point to add attributes that are needed to fully represent resources in the case that a system contains more additional data. Examples of such plug-ins attributes are the optional attributes "Availability", "Performance", and "Interface" discussed in Chapter 3. For this reason H6 is located in the class "Resources" with other basic resources attributes as seen in Figure 5.1.

The Hook H6 only accepts the insertion of plug-ins of type ATTR and integrates them with other basic resources attributes.

5.3.2.7 Characteristics of Hook H7

Hook H7 is an insertion point to add steps that might need to be performed on the resources. For this reason H7 is located in the class "Resources" with other resources steps as seen in Figure 5.1.

Hook H7 only accept the insertion of plug-ins of type STEP and adds them with other resources steps.

5.3.2.8 Characteristics of Hook H8

Hook H8 is an insertion point to add interactions scenarios that the analyst might use to achieve more thoroughly detected interactions. For this reason H8 is located in the class "Interactions Detection" with other IRIS basic core interaction scenarios as in Figure 5.1. Hook H8 only accepts the insertion of plug-ins of type SCR (where SCR stands for scenario) and adds them with other basic core interaction scenarios.

5.4 IRIS Plug-ins

5.4.1 General Structure of a Plug-in

A critical point when creating plug-ins for IRIS is to follow the general format and structure of plug-ins to ensure their integrity.

The general structure of a plug-in to be used with IRIS 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 the location where this plug-in can be hooked to the basic core of IRIS. Figure 5.2 shows a description of the general structure of an IRIS plug-in.

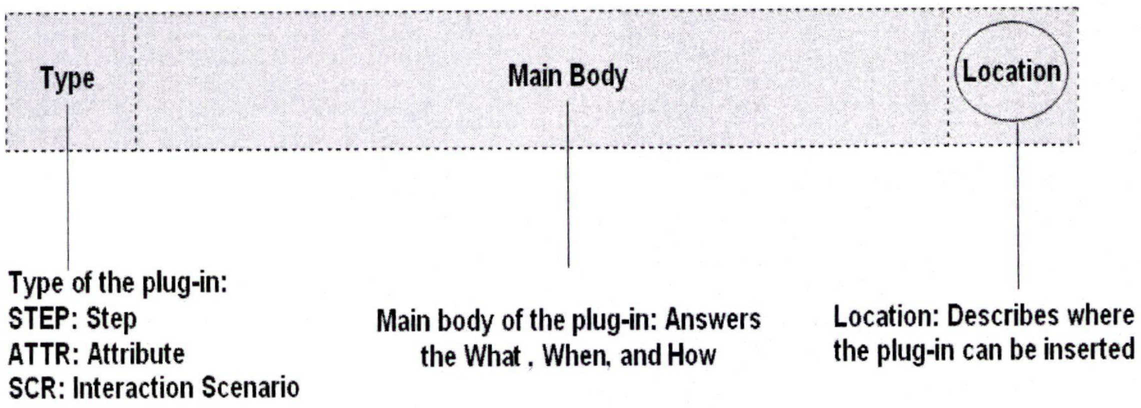

Type
Main Body
Location
Type of the plug-in:
STEP: Step
ATTR: Attribute
SCR: Interaction Scenario
Main body of the plug-in: Answers the What , When, and How
Location: Describes where the plug-in can be inserted

Figure 5.2: General structure of an IRIS plug-in

5.4.2 Plug-in Type

As shown in Figure 5.2, the Type is a three or four letter abbreviation that describes the type of the plug-in. A plug-in can have one of the following types:

- Step (STEP): A STEP plug-in is an independent step that generates its own set of tables and graphs. This type of plug-in is needed when there is a certain step that is not necessarily always applied, like representing requirements in a graphical notation to make sure that analyst understands how each requirement behaves.
- Attribute (ATTR): An attribute is used to describe a specific part of a requirement (see Chapter 3). For example, in the smart homes domain (presented in Chapter 8), many requirements have parameters in their body and therefore the two attributes "Parameters" and "Parameters Range" have been inserted as plug-ins into IRIS to analyze the smart homes domain.
- Interaction Scenario (SCR): An interaction scenario is a description of a situation in which two requirements interact and how this interaction can be detected by a human analyst.

5.4.3 Plug-in Main Body

The second part of a plug-in is the plug-in main body. The plug-in main body describes what this plug-in is and when and how the analyst should use it. The plug-in body has the following parts:

- What: states what this plug-in is
  - o Name: A unique descriptive name of the plug-in
  - o Description: A textual description of what this plug-in is
  - o Construction: The internal construction of the 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
    - Expected enhancement: A description of the expected enhancement this plug-in will provide
- How: states how to apply this plug-in
    - Instructions: A set of instructions on how to insert this plug-in plus any other instructions
    - Example of application: A sample description of how to use the plug-in

5.4.4 Plug-in Location

The "Location", as shown in Figure 5.2, describes where this plug-in can be inserted. For instance, if a plug-in is inserted into hook H2, the "Location" of the plug-in is assigned the value H2. This prevents any mis-location of plug-ins.

5.4.5 Available Plug-ins for IRIS

So far 10 plug-ins have been designed that have the structure described above and are fully documented. These plug-ins were identified and designed based on the case studies carried out in this thesis and also based on the need to add optional attributes to the system elements (e.g., adding the attributes parameters and parameters range to system axioms or dynamic behaviour requirements). However, additional plug-ins can be designed in the future by the author or by other developers if needed.

As an example, Table 5.1 presents a full description of the plug-in named "Graphical representation of individual requirements". It is worth mentioning that the interaction scenario plug-in (SCR) is not described as it is fully detailed in Chapter 3 and Appendix

B. The remaining 8 plug-ins are briefly described below. However, Appendix D presents the full details of each of the 8 Plug-ins using the structure described in subsection 5.4.3.

Table 5.1: 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. This is to ensure that the analyst fully understands the behaviour of the requirements.
		Construction	The execution of this step requires the following activities: 1. Select every requirement from the set of given requirements, list it separately, and read it carefully. 2. Identify a suitable graphical representation (e.g., UML notations [143-146], CRESS [65], UCM[147-149]),. 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 4. Improved interaction detection and prevention of false interactions
	How	**Instructions**	1. This step is applied prior to step 1 of IRIS basic core.
		Sample of application	This step has been applied in a case study to identify interactions between the requirements of a lift system. Refer to Chapter 6 for an example application.
Location	Since this is a STEP plug-in that is needed to be performed prior to the application of IRIS basic core steps, then this step is hooked to the hook H1		

- *Parameter Assignment*: This is a STEP plug-in and is used to find any parameterized parts in the given set of requirements. These parameterized parts are then 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 when night starts". The "Certain place" is a parameterized part and the Parameter Assignment plug-in replaces this part with the parameter X. The requirement now reads "the lights will switch on in place X when the night starts". Since this is a STEP plug-in that is needed to be performed prior to the application of IRIS basic core steps, this plug-in is hooked into the hook H1.
- *Parameters*: This is an attribute (ATTR) plug-in. It corresponds to adding the attribute "Parameters" to the set of attributes used for representing system axioms requirements or dynamic behaviour requirements. The use of this plug-in results in a new column, called "Parameters", in the tables created for the system axioms or the dynamic behaviour requirements. This new column contains the different parameters used in each requirement along with the data type allowed for these parameters. This plug-in must be used in conjunction with the *"Parameter assignment"* plug-in. Since this is an ATTR plug-in that is needed to add the attribute Parameters to either system axioms or dynamic behaviour, then this plug-in is hooked into the hooks H2 or H4.
- *Parameters Range*: This is an attribute plug-in that has to be used in conjunction with the Parameters plug-in. It corresponds to adding the attribute "Parameters Range" to the set of attributes used for representing system axioms requirements or dynamic behaviour requirements. The use of this plug-in results in a new

column, called "Parameters range", in the tables created for the system axioms or the dynamic behaviour requirements. The new column describes the allowed range of values that each parameter can have. Since this is an ATTR plug-in that is needed to add the attribute Parameters Range to either system axioms or dynamic behaviour, this plug-in is hooked into the hooks H2 or H4.

- *Functionalities identification*: This is a STEP plug-in that is used when a single requirement is complex and describes different functionalities. For example a requirement for an intruder alarm has many functionalities within the same requirement. The goal of this plug-in is to simplify the parent requirement by breaking it down into atomic functionalities that can be easily handled. Since this is a STEP plug-in is needed prior to the application of IRIS basic core steps, this plug-in is hooked into the hook H1.
- *Graphical representation of individual requirements*: This is a STEP plug-in that corresponds to a complete step that is performed prior to the application of IRIS basic core. This plug-in is used when the given set of requirements are vague and therefore must be fully understood before proceeding with the remaining IRIS steps. A complete description of this plug-in was given in Table 5.1.
- *System axioms strategies*: This is a STEP plug-in, i.e., a new step is carried out to identify the different strategies used for the design and implementation of system axioms. This plug-in creates a table to describe the system axioms design and implementation strategies. Since this is a STEP plug-in that performs a certain step on the system axioms, this plug-in is hooked into the hook H3.

- *Availability*: This is an attribute (ATTR) plug-in. It corresponds to adding the attribute "Availability" to the set of attributes used for representing resources requirements. The use of this plug-in results in a new column in the table created for the resources requirements. The new column contains the values of the availability of each resource requirement. Since this is an ATTR plug-in that adds the attribute "Availability" to the resources, this plug-in is hooked into the hook H6.
- *Performance*: This is an attribute (ATTR) plug-in. It corresponds to adding the attribute "Performance" to the set of attributes used for representing resources requirements. The use of this plug-in results in a new column in the table created for the resources requirements. The new column contains the values of the performance of each resource requirement. Since this is an ATTR plug-in that adds the attribute Performance to resources, this plug-in is hooked to the hook H6.
- *Interface*: This is an attribute (ATTR) plug-in. It corresponds to adding the attribute "Interface" to the set of attributes used for representing resource requirements. The use of this plug-in results in a new column in the table created for the resources requirements. The new column contains the values regarding the interface for each resource requirement. Since this is an ATTR plug-in that adds the attribute Interface to resources, this plug-in is hooked into the hook H6.
- *SCRi*: The SCRi corresponds to the ith interaction scenario (SCR) plug-in. This plug-in can correspond to the following plug-ins interaction scenarios: SCR5, SCR6, SCR7, SCR9, SCR14, SCR15, SCR16, SCR17, SCR18, SCR19, SCR20, SCR21, SCR22, SCR23, SCR24, SCR25, SCR26, SCR27, SCR28, SCR29,

SCR32, SCR33, SCR34, SCR35, SCR36, and SCR37. The details of each of these interaction scenarios are described in details in Chapter 3 and Appendix B. These interaction scenarios are used to increase the thoroughness for detecting interactions between requirements. However, it is worth saying that some interaction scenarios plug-ins cannot be used unless other plug-ins are used. For example, all the interaction scenarios plug-ins (SCR17, SCR18, SCR22, SC23, SCR27, SCR32, and SCR35) which are aimed at detecting interactions due to a requirement resource availability attribute, cannot be used unless the plug-in "Availability" has been hooked to IRIS and is being used. Since this is an SCR plug-in that is needed to add interaction scenarios to "Interactions Detection", this plug-in is hooked into the hook H8.

5.5 Summary

This Chapter presented the customization of IRIS to detect interactions in any domain and at different levels of thoroughness. To achieve such a goal, IRIS was designed with a basic core as well as extension hooks for expansion through the addition of plug-ins that can be attached to the hooks.

The plug-ins can be used to ensure the successful application of IRIS in new domains and also enhance the interaction detection results by providing more steps, tables, interaction scenarios…etc to detect interactions more thoroughly. Currently 10 plug-ins have been created in this thesis for extending and enhancing the performance of IRIS as needed. However, as a powerful feature, new plug-ins can be developed by analysts who are using IRIS to accommodate any special needs and hence to successfully apply IRIS to detect interactions. When creating new plug-ins, an analyst must follow the general structure of plug-ins to ensure the integrity of the approach and hence its successful application to detect interactions. The next chapters describe the application of IRIS to detect interactions in different domains using its basic core and some of the plug-ins described in this chapter.

CHAPTER SIX: APPLYING IRIS IN THE CONTROL DOMAIN - THE LIFT SYSTEM CASE STUDY

6.1 Introduction

The lift system is a well recognized system from the control domain that is often used as a benchmark for validating new approaches for interaction detection. This chapter presents the application of the proposed semi-formal approach IRIS to detect interactions in the lift system. The lift system case study consists of a set of 14 requirements that describes the basic operation of a simple lift system. Hence, IRIS is applied to detect interactions in this case study at the requirements level.

IRIS was able to detect 7 interactions between the lift system requirements. The results were compared with the results reported by Heisel *et al.* in [18, 150]. IRIS was able to detect all interactions reported by Heisel *et al.* in [18, 150]. IRIS was also able to detect an interaction that [18, 150] did not detect. Moreover, IRIS achieved a 17.6% reduction in the number of comparisons that a human expert would have to do to compare all 14 requirements.

This chapter is structured as follows: Section 6.2 presents the requirements of the lift system that were used in the case study. Section 6.3 shows how IRIS was customized to more effectively detect interactions in the lift system. Section 6.4 describes in a step-by-step manner the application of IRIS to the lift system requirements along with the results obtained from each step. Section 6.5 contains a discussion and a comparison of the obtained results with the results reported by Heisel *et al.* in [18, 150]. Finally, in section 6.6 a summary of the chapter is presented.

6.2 The Lift System Requirements

In the lift system case study, the following14 requirements describing the basic behaviour of a simple lift have been identified [18, 150]:

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 an 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.

6.3 Customizing IRIS for the Lift System Case Study

6.3.1 Plug-ins used in the Lift System Case Study

To illustrate what plug-ins have been used in the lift system case study, a list of the problems encountered in this case study is described first. Then the plug-ins that were used to overcome these problems are described.

1. The initial textual description of the lift requirements was unclear and some requirements did not provide a clear understanding on how the system should behave when these requirements are triggered (e.g., R3 and R14). The plug-in "*Graphical representation of individual requirements*" was used to graphically represent requirements and understand their exact behaviour. This helped resolve the ambiguities that existed earlier on by visually modeling these requirements. This plug-in is hooked to hook H1 and therefore is carried out prior to the execution of step 1 of the basic core of IRIS.
2. To detect all possible interactions between a system axiom and a dynamic behaviour requirement, the interaction scenarios "SCR5", "SCR6", and "SCR7" were inserted into IRIS as plug-ins at hook H8 and were used to provide interaction detection between system axioms and dynamic behaviour requirements. These interaction scenarios are applied as part of sixth step of IRIS.
3. To detect all possible interactions between two dynamic behaviour requirements, four interaction scenarios have been inserted into IRIS as plug-ins at hook H8, namely: SCR9, SCR14, SCR15, and SCR16. These interaction scenarios are applied as part of the sixth step of IRIS.

6.3.2 Assumptions used in the Lift System Case Study

1. The lift system is described by a set of 14 requirements. Hence IRIS was applied to detect interactions at the requirements level.
2. The set of 14 requirements were chosen as they explained the basic operation of a simple lift system. Other requirements such as "Executive floor" or "Multi-Car" were not included in this case study for simplicity purpose.

6.4 Applying IRIS to Detect Interactions in the Lift System Case Study

This section presents the application of IRIS to the lift system requirements presented in Section 6.2. The basic core steps of IRIS as well as the plug-ins used in this case study are presented in the order of their execution.

6.4.1 Using the Plug-in "Graphical representation of individual requirements"

The plug-in "*Graphical representation of individual requirements*" used the CRESS notation [65] to graphically represent ambiguous requirements. Figure 6.1 shows a sample of using CRESS to represent requirements R1 and R3.

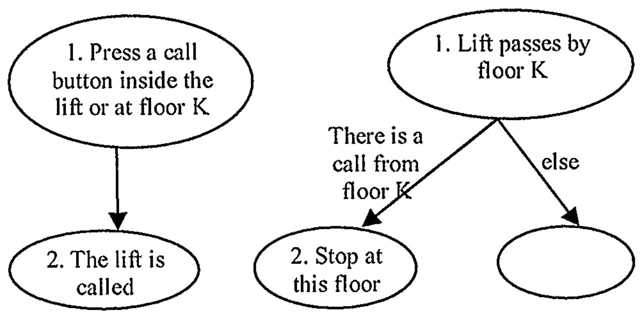

1. Press a call button inside the lift or at floor K
2. The lift is called
1. Lift passes by floor K
There is a call from floor K
else
2. Stop at this floor

Figure 6.1 CRESS [65] representation for R1 on the left and R3 on the right

Figure 6.1 shows that if the lift passes by floor K and there is a call from floor K, the lift will stop at this floor. If there is no call the lift will proceed with normal operation and no action is taken (represented by the empty oval on the right hand side of Figure 6.1).

6.4.2 Step 1: Requirements Classification

After analyzing the lift system requirements, they are classified into system axioms and dynamic behaviour requirements as shown in Table 6.1.

Table 6.1: Classification table for the lift system requirements

System Axioms	R2, R6, R7, R8, and R11
Dynamic Behaviour Requirements	R1, R3, R4, R5, R9, R10, R12, R13, and R14

6.4.3 Step 2: Requirements Attributes Identification

Table 6.2 contains the values of the different attributes of each system axiom, and Table 6.3 contains the values of the different attributes of each dynamic behaviour requirement.

Table 6.2: System axioms attributes identification table for the lift system,

ID	Description	Rule	Condition
R2	Pressing a call button is possible at any time.	Pressing any button is always available to the user	True
R6	The lift only changes its direction when there are no more calls in the current direction.	Changing direction is possible	No more calls in the current direction
R7	When there are no more calls, the lift stays at the floor last served.	Lift stays at floor last served	No more calls
R8	As long as there are unserved calls, the lift will serve these calls.	The lift will always serve unserved calls	There are unserved calls
R11	Whenever the lift moves, the doors must be closed.	Doors are closed	Lift is moving

Table 6.3: Dynamic behaviour attributes identification table for the lift system

ID	Description	Pre-State	Trigger Event	Action	Next State
R1	The lift is called by pressing a call button, either at a floor or inside the lift.	Lift not called	Pressing a call button	Call the lift	Lift is called
R3	When the lift passes by floor K, and there is a call for this floor, the lift will stop at floor K.	Lift is moving	Lift passes by floor K AND Call from floor K	Stop at floor K	Stopped at floor K
R4	When the lift has stopped, it will open the doors.	Lift is moving	Lift has stopped	Open the doors	Doors opened
R5	When the lift doors have been opened, they will close automatically after d time units.	Doors are opened	Doors have finished opening AND d time units have elapsed	Close the doors automatically	Doors closed
R9	When the lift is halted at floor K with the doors opened, a call from floor K is not taken into account.	At floor K with doors opened	Call from floor K	Ignore call from this floor	At floor K with doors opened
R10	When the lift is halted at floor K with doors closed and receives a call from floor K, it reopens its doors.	At floor K with doors closed	Call from floor K	Reopen the doors	Doors are opened
R12	The closing of a door may be prevented by pressing an open-door button.	Doors are closing	Pressing open door button	Prevent doors closing	Doors are opened
R13	When something blocks the doors, the lift interrupts the process of closing the door and reopens the doors.	Doors are closing	Something blocks the doors	interrupt door closing and reopens doors	Doors are opened
R14	When the lift is overloaded, doors will not close.	Doors are opened	Lift is overloaded	Do not close the doors	Doors are opened

6.4.4 Step 3: Trigger Events Extraction

After analyzing the triggers required to trigger the dynamic behaviour requirements of the lift system, 9 trigger events are extracted and identified as shown in Table 6.4.

Table 6.4: Trigger events extraction table for the lift system

Event ID	Event Description	Requirements Triggered by this Event
E1	Pressing a call button	R1
E2	Call from floor K	R3, R9, R10
E3	Lift passes by floor K	R3
E4	Lift has stopped	R4
E5	Doors have finished opening	R5
E6	Pressing open door button	R12
E7	Something blocks the doors	R13
E8	d time units have elapsed	R5
E9	Lift is overloaded	R14

6.4.5 Step 4: Linked Events Identification

Table 6.5 shows the results of identifying linked events (step 4). It must be noted that the event number does not imply the direction of the link as can be seen with E6 which is linked to E5.

Table 6.5: Linked events identification table for the lift system

Event ID	Event Description	Linked to	Mathematical Representation
E1	Pressing a call button	E2, E4	E1 ~> E2, E1 ~> E4
E2	Call from floor K	E4	E2 ~> E4
E3	Lift passes by floor K	E4	E3 ~> E4
E4	Lift has stopped	E5	E4 ~> E5
E5	Doors have finished opening	E7, E9	E5 ~> E7, E5 ~> E9
E6	Pressing open door button	E5, E9	E6 ~> E5, E6 ~> E9
E8	d time units have elapsed	Ei, i=1, 2, 3, 4, 5, 6, 7, 9	E8 ~> Ei, i=1, 2, 3, 4, 5, 6, 7, 9
E9	Lift is overloaded	E4, E8	E9 ~> E4, E9 ~> E8

6.4.6 Step 5: Trigger Events Charts Representation

Figure 6.2 shows the 9 trigger events and the requirements they trigger using trigger events charts. It is worth mentioning that some requirements need to be triggered by more than one trigger event in order to execute (e.g., R3). In this case, the extra trigger events are represented in the state charts in the form of logical AND constraints which are represented by the symbol (|||) (e.g., E3 and E2 triggering R3 at the top right hand side of Figure 6.2).

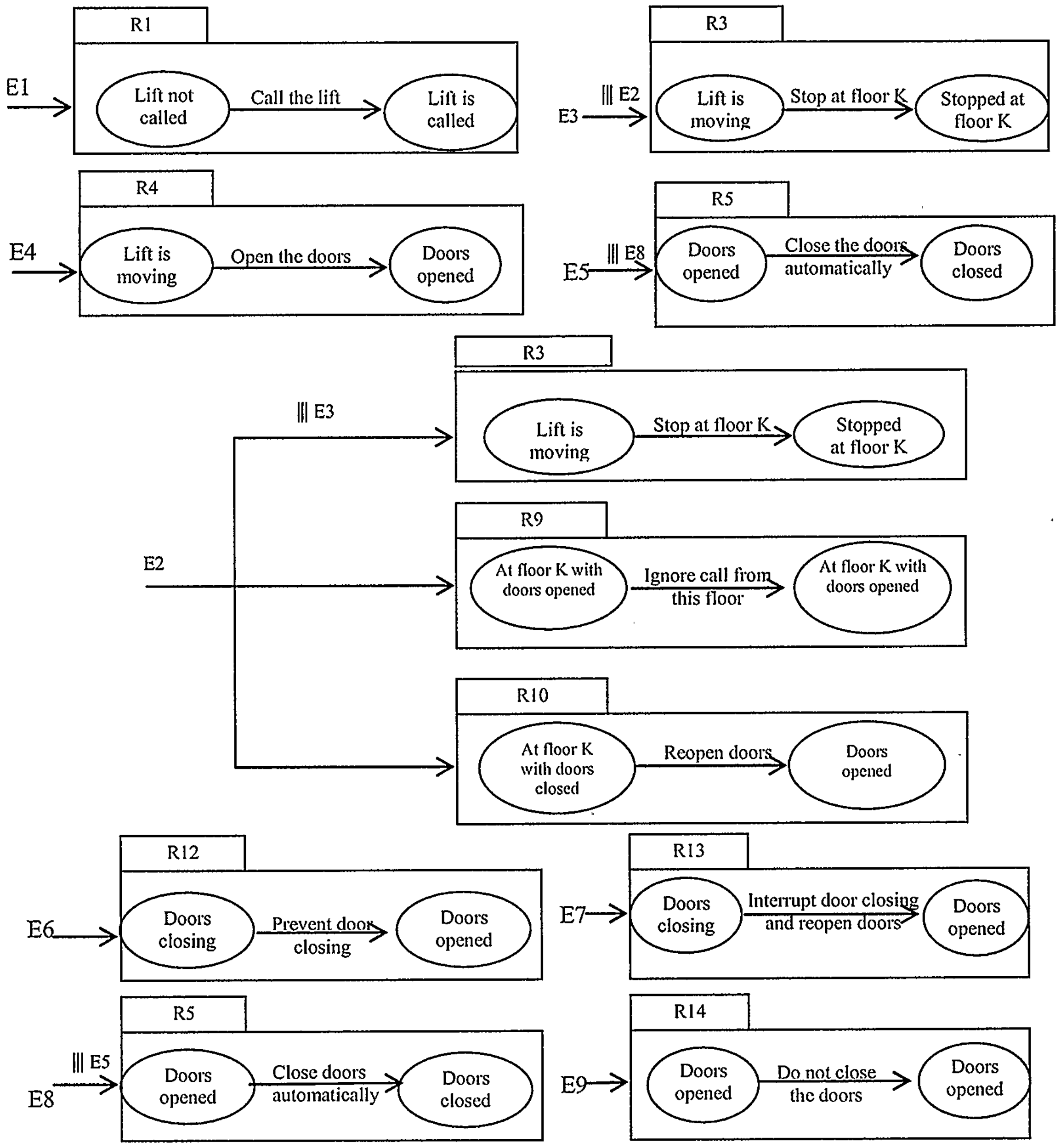


Figure 6.2 Trigger events charts for the dynamic requirements of the lift system

6.4.7 Step 6: Interaction Detection

6.4.7.1 Summary of the Detected Interactions

In this step, the developer detects interactions between requirements using interaction scenarios that are either within the basic core of IRIS or interaction scenarios that are inserted as plug-ins into IRIS. The detection is subjective which means that a developer uses the different tables and graphs developed along with the provided interaction scenarios to determine if there exists any interaction between two requirements.

The developer now tries to find interactions as explained in Section 4.3.7. Table 6.6 provides a summary of the detected interactions in the lift system case study. However, an illustration is given below on how these interactions were detected in the lift system case study.

Table 6.6: A summary of the detected interactions in the lift system case study

Requirement	Interacting Requirements
R9	R1
R12	R5 and R8
R13	R5 and R8
R14	R5 and R8

6.4.7.2 Interactions According to Main Interaction Category ①

In this interaction category, two interaction scenarios are used, namely: SCR1 and SCR2. The developer has to pair-wise compare all system axioms with the aim of finding interactions according to SCR1 and SCR2.

The analysis of the system axioms of the lift system using SCR1 and SCR2 did not result in any detected interactions.

6.4.7.3 Interactions According to Main Interaction Categories ② and ⑦

Seven interaction scenarios are used under interaction categories ② and ⑦, namely: SCR3, SCR4, SCR5, SCR6, SCR7, SCR 30, and SCR31. The developer is required to examine the system axioms attributes identification table (Table 6.2) and the dynamic behaviour requirements attributes identification table (Table 6.3). The developer has to perform pair-wise comparison of every system axiom and every dynamic behaviour requirement with the objective of finding interactions according to the seven interactions scenarios.

Three interactions were detected using the interaction scenario SCR30. The three detected interactions are: interaction between R12 and R8, interaction between R13 and R8, and interaction between R14 and R8. The details of these interactions are described in Tables 6.7, 6.8, and 6.9, respectively.

Table 6.7: Interaction between R12 and R8 in the lift system case study

Interaction ID	I5
Type of Interaction	Interaction between a dynamic behaviour requirement and a system axiom
Interacting Requirements	R12 and R8
Interaction Scenario used	SCR30
Explanation of Interaction	There is a contradiction between the value of the *Action* attribute of the dynamic behaviour requirement (R12) and the value of the *Rule* attribute for the system axiom (R8). The action of R12 will override the rule of R8. A possible interaction situation could be the following: A user keeps pressing the open door button for a long time and hence the lift is unable to serve other unserved calls.

Table 6.8: Interaction between R13 and R8 in the lift system case study

Interaction ID	I6
Type of Interaction	Interaction between a dynamic behaviour requirement and a system axiom
Interacting Requirements	R13 and R8
Interaction Scenario used	SCR30
Explanation of Interaction	There is a contradiction between the value of the *Action* attribute of the dynamic behaviour requirement (R13) and the value of the *Rule* attribute for the system axiom (R8). The action of R13 will override the rule of R8. A possible interaction situation could be the following: A user puts anything like a rock to block the process of closing the doors and hence the lift doors are always kept open and hence the lift is unable to serve other unserved calls.

Table 6.9: Interaction between R13 and R8 in the lift system case study

Interaction ID	I7
Type of Interaction	Interaction between a dynamic behaviour requirement and a system axiom
Interacting Requirements	R14 and R8
Interaction Scenario used	SCR30
Explanation of Interaction	There is a contradiction between the value of the *Action* attribute of the dynamic behaviour requirement (R14) and the value of the *Rule* attribute for the system axiom (R8). The action of R14 will override the rule of R8. A possible interaction situation could be the following: A user is using the lift to move furniture and puts many things which overload the lift. Consequently the lift doors will not close and will remain open. If that user does not remove some furniture out, then the lift doors are kept open and will not be able to serve other unserved calls.

6.4.7.4 Interactions According to Main Interaction Category ③

The basic core of IRIS contains five interaction scenarios to be used for detecting interactions between two dynamic behaviour requirements: SCR8, SCR10, SCR11, SCR12, and SCR13. However, additionally four plug-ins interaction scenarios were inserted and used which are: SCR9, SCR14, SCR15, and SCR16.

To detect interactions between two dynamic behaviour requirements, the analyst first looks at trigger events charts in Figure 6.2 and the linked events table shown in Table 6.5 and identifies unique pairs of requirements that are triggered by the same trigger event or by linked trigger events. This resulted in the following pairs of requirements:

S(R3, R9), S(R3, R10), S(R9, R10), L(R1, R3), L(R1, R9), L(R1, R10), L(R1, R4), L(R3, R4), L(R9, R4), L(R10, R4), DL(R4, R5), L(R5, R13), DL(R5, R12), L(R12, R14), L(R5, R1), L(R5, R3), L(R5, R9), L(R5, R10), L(R14, R4),and DL(R5, R14).

The following symbols have been used to describe the pairs of requirements:

- S(R_i, R_j): The two requirements R_i and R_j are triggered by the same trigger event
- L(R_i, R_j): The two requirements R_i and R_j are triggered by linked trigger events such that E_i ~> E_j
- DL(R_i, R_j): The two requirements R_i and R_j are sequentially related through E_i ~> E_j and also they are sequentially related through E_j ~> E_i (called dual linked events)

Now, the analyst has to analyze the requirements pairs listed above using the 9 interaction scenarios. This analysis resulted in the following interactions to be detected: interaction between R12 and R5, interaction between R13 and R5, interaction between R14 and R5, and interaction between R9 and R1. The details of these interactions are described in Tables 6.10-6.13, respectively.

Table 6.10: Interaction between R12 and R5 in the lift system case study

Interaction ID	I1
Type of Interaction	Interaction between two dynamic behaviour requirements
Interacting Requirements	R12 and R5
Interaction Scenario used	SCR12
Explanation of Interaction	There is a contradiction between the value of the *Action* attribute of the dynamic behaviour requirement (R12) and the value of the of the *Action* attribute of the dynamic behaviour requirement (R5). The action of R12 will override the action of R5. A possible interaction situation could be the following: After the doors are opened, a user keeps pressing the open door button for a long time and hence the lift doors are unable to close after d time units.

Table 6.11: Interaction between R13 and R5 in the lift system case study

Interaction ID	I2
Type of Interaction	Interaction between two dynamic behaviour requirements
Interacting Requirements	R13 and R5
Interaction Scenario used	SCR12
Explanation of Interaction	There is a contradiction between the value of the *Action* attribute of the dynamic behaviour requirement (R13) and the value of the of the *Action* attribute of the dynamic behaviour requirement (R5). The action of R13 will override the action of R5. A possible interaction situation could be the following: After the doors are opened, a user puts anything like a rock to block the process of closing the doors and hence the lift doors are always kept open and hence the lift doors are unable to close after d time units.

Table 6.12: Interaction between R14 and R5 in the lift system case study

Interaction ID	I3
Type of Interaction	Interaction between two dynamic behaviour requirements
Interacting Requirements	R14 and R5
Interaction Scenario used	SCR12
Explanation of Interaction	There is a contradiction between the value of the *Action* attribute of the dynamic behaviour requirement (R14) and the value of the of the *Action* attribute of the dynamic behaviour requirement (R5). The action of R14 will override the action of R5. A possible interaction situation could be the following: After the doors are opened, a user uses the lift to move furniture and puts many things which overload the lift. Consequently the lift doors will not close and will remain open. If that user does not remove some furniture out, then the lift doors are kept open and hence the lift doors are unable to close after d time units.

Table 6.13: Interaction between R9 and R1 in the lift system case study

Interaction ID	I4
Type of Interaction	Interaction between two dynamic behaviour requirements
Interacting Requirements	R9 and R1
Interaction Scenario used	SCR12
Explanation of Interaction	There is a contradiction between the value of the *Action* attribute of the dynamic behaviour requirement (R9) and the value of the of the *Action* attribute of the dynamic behaviour requirement (R1). The action of R9 will override the action of R1. A possible interaction situation could be the following: The lift is at floor K with doors opened and are about to close the doors in less than a second. Someone outside the lift system presses the call button to call the lift. According to R1, the lift should be called and give him sufficient time to ride the lift. However, according to R9, which will override the action of R1. The call is ignored because the lift is at floor k with its doors opened, and hence the call from this floor is ignored and the doors start closing not giving the user, who pressed the call button, sufficient time to ride the lift.

6.5 Discussion of the Results

6.5.1 Reduction in Number of Comparisons

IRIS can reduce the number of comparisons that needs to be performed by an expert to informally detect interactions between the given set of requirements.

In the lift system case study, IRIS needed to perform 75 pair-wise comparisons as follows:

- 10 comparisons to detect interactions according to main interaction category ① (number of all possible pair-wise comparisons according to Table 6.2)
- 45 comparisons to detect interactions according to main interaction categories ② and ⑦ (number of all possible pair-wise comparisons according to Table 6.2 and Table 6.3)

- 20 comparisons to detect interactions according to main interaction category ③ (number of comparisons between two dynamic behaviour requirements triggered by the same event or linked events as explained in Section 6.4.7.4)

If a human expert would have to pair-wise compare all of the lift system requirements informally, s/he would have needed 91 comparisons. This means that IRIS has achieved a 17.6% reduction in number of comparisons.

Although this 17.6% cannot be translated into the same percentage reduction of time and effort due to the overhead associated with applying IRIS, it still shows that there is a reduction in time and effort especially when an IRIS-trained developer conducts the case study and the number of requirements is high.

6.5.2 Comparing IRIS Results with the Results by Heisel *et al.* in [18, 150]

In section 6.4, IRIS was applied to detect interactions at the requirements level between 14 requirements of the lift system. The case study had the following numbers:

Number of Requirements	14
Number of detected interactions using IRIS	7
Number of performed comparisons using IRIS	75
Number of comparisons an expert would have to do to compare all requirements	91

To discuss and evaluate the obtained results, we compare them with the results by Heisel *et al.* in [18, 150].Heisel *et al.* [18, 150] reported results on detecting interactions between requirements of the lift system. In [18, 150], Heisel *et al.* have detected 6 interactions between the 14 requirements of the lift system versus 7 interactions that were detected using IRIS. IRIS was not only able to detect all the interactions reported by Heisel *et al.* it also found an interaction between R14 and R8 which was missed by Heisel *et al.* IRIS was able to detect this additional interaction as it analyses system axioms and dynamic behaviour requirements with human involvement which the approach by Heisel *et al.* [18, 150] lacks.

6.6 Summary

This chapter presented the application of the IRIS to the lift system case study from the control domain. In general, the lift system had a set of 14 requirements. IRIS was successful in detecting 7 interactions between the lift system requirements. To examine the accuracy of the detected interactions, IRIS was compared to the results reported by Heisel *et al.* in [18, 150]. IRIS was able to detect all the interactions that are reported in literature by Heisel *et al.* [18, 150] and found an additional interaction between R14 and R8 which the approaches described in [18, 150] failed to detect. Moreover, IRIS achieved a 17.6% reduction in the number of comparisons that an expert would have to perform to compare all the 14 requirements of the lift system which indicates reduction in time and effort.

CHAPTER SEVEN: APPLYING IRIS IN THE TELECOMMUNICATIONS DOMAIN - THE TELEPHONY FEATURES CASE STUDY

7.1 Introduction

This chapter presents the application of IRIS in the telecommunications domain. This case study was conducted using a set of 8 telephony features that were provided by the feature interaction contest held in 2000 [19]. The 8 telephony features are implemented on top of the Plain Old Telephony System (POTS) [151]. IRIS was applied to detect interactions between the 8 telephony features and hence IRIS is applied to detect interactions at the features level.

IRIS was able to detect 21 interactions in this case study. To validate these results, a comparison is made with other results reported by researchers using different approaches in the Second Feature Interaction Contest held in 2000 (FIW00) [19]. Moreover, IRIS achieved a 17.9% reduction in the number of comparisons that a human expert would have to do to compare all 8 telephony features.

The structure of this chapter is as follows: Section 7.2 presents a description of the 8 telephony features used in the case study. Section 7.3 shows how IRIS was customized to be applied in the telephony features case study. Section 7.4 shows the application of IRIS to detect interactions among the 8 telephony features along with the results obtained from each step of IRIS. Section 7.5 presents a discussion of the obtained results along with a comparison of these results with other results reported in the FIW00 contest. Finally, Section 7.6 presents the chapter summary.

7.2 The Telephony Features

The second feature interaction contest was held in conjunction with the Sixth International Workshop on Feature Interaction in Telecommunications and Software Systems (FIW00) [19]. In this case study a set of 8 features given in the contest is used for interaction detection with IRIS as shown in Table 7.1.

Table 7.1: A description of the telephony features used in the case study

Feature Name	Feature Abbreviation	Feature Informal Definition
Call Forward on Busy Line	CFBL	All calls to a subscriber line are redirected to a predefined number when the subscriber line is busy.
Teen Line	TL	During a pre-set time of the day, this feature restricts all outgoing calls from the subscriber's telephone unless a PIN is provided.
Terminate call Screening	TCS	All incoming calls to the subscriber's telephone are screened against a screening list. If the originator of an incoming call matches an entry in the list, the call is terminated.
Call Waiting	CW	This feature allows the subscriber to be notified of an incoming call while s/he is busy and to accept the new call by putting the original call on hold. Then s/he is able to toggle between the two calls.
Three Way Calling	3WC	This feature allows a user already connected to another user to bring a third party into the call. The subscriber can setup a connection to the new party by putting the current partner on hold, connecting to the third party and joining lines. The 3WC is terminated by any side going on hook.
Reverse Charge	RC	Allows the subscriber to be charged for all calls in which the subscriber is the terminating party.
Ring Back when Free	RBF	When a call attempt is made to a busy line with this feature active, the caller is informed that s/he will be called back when the other person is free. Once the subscriber terminates his/her call, a connection to the stored numbers will be established.
Voice Mail	VM	This offers the possibility to leave a message if the called party is busy or not answering.

7.3 Customizing IRIS for the Telephony Features Case Study

7.3.1 Plug-ins used in the Telephony Features Case Study

1. The descriptions of the features provided by the contest organizers for the 2nd feature interaction contest [19] were very detailed and addressed all the questions that might be asked about the behaviour and design of the features. Therefore, only interaction scenarios plug-ins were inserted into IRIS at hook H8, and are used to provide thorough interaction detection. Since all features are dynamic behaviour features, four plug-ins interaction scenarios have been used: SCR9, SCR14, SCR15, and SCR16. These plug-ins are applied as part of IRIS step 6.

7.3.2 Assumptions used in the Telephony Features Case Study

1. IRIS is being applied to detect interactions at the features level.
2. Informal definitions of the telecommunications features are used. Low-level design or implementation details are not considered in this case study as they are beyond the scope of IRIS (See section 4.5 regarding the limitations of IRIS).

7.4 Applying IRIS to Detect Interactions in the Telephony Features Case Study

7.4.1 Step 1: Features Classification

In the first step the features are organized into system axioms or dynamic behaviour features. Since all the features in this case study describe the dynamic behaviour of the system, they are classified as dynamic behaviour features.

7.4.2 Step 2: Features Attributes Identification

In this step, each dynamic behaviour feature is analyzed to identify the values of its attributes. Table 7.2 presents the values of the attributes of the dynamic behaviour features used in this case study.

Table 7.2: Dynamic behaviour attributes identification for telephony features

ID	Description	Pre-state	Trigger Event	Action	Next state
CFBL	All Calls to a subscriber Line are redirected to a predefined number when the subscriber line is busy.	Busy	Call request	Redirect the incoming call to a predefined number	Busy
TL	During a pre-set time of the day, this feature restricts all outgoing calls from the subscriber's telephone unless a PIN is provided	Idle ||| Time T in restricted time zone	Call attempt	Ask for PIN. If the PIN is ok connect otherwise disconnect	busy or idle
TCS	The originators of all incoming calls to subscriber's telephone are screened against a screening list. If the originator of an incoming call matches an entry in the list, the call is terminated.	Idle	Call request ||| calling party is matching an entry in TCS list	Terminate call	Idle
CW	This feature allows the subscriber to be notified of an incoming call while busy and to accept the new call by putting the original call on hold. He is able to toggle between the two calls.	Busy	Call request	The user can accept the new call putting the original on hold then he can toggle between them.	Busy
3WC	This feature allows a user already connected to another user to bring a third party into the call. The subscriber can setup a connection to the new party by putting the current partner on hold, connecting to the third side and joining lines. The 3WC is terminated by any side going on hook.	Busy	Flash signal ||| call attempt	Connect to the third party and then join both calls	Busy
RC	Allows the subscriber to be charged for all calls in which the subscriber is the terminating party.	Idle	Call request ||| called party answer	Charge called party	Busy
RBF	When a call attempt is made to a busy line with this feature active, the caller is informed that he will be called back when the other person is free. Once the subscriber terminates his call, a connection to the stored numbers is established.	Busy	Call request	Store number and automatically call it back when phone is free	Busy
VM	This offers the possibility to leave a message if the called party is busy or not answering.	Idle	(Call request) ||| NOT (called party answer)	Allow the caller to leave a message	Idle
		busy	Call request		busy
Where ||| represents a logical AND					

7.4.3 Step 3: Trigger Events Extraction

In this step the developer identifies and extracts all the different trigger events of the dynamic behaviour features listed in Table 7.2. The output of this step resulted in a list of 5 different trigger events which are listed in Table 7.3. Note that a call attempt indicates a user initiating a phone call while call request indicates a user receiving a phone call.

Table 7.3: Trigger events extraction table for the telephony features case study

Event ID	Event Description	Features Triggered by this Event
E1	Call request	CFBL, TCS, CW, RC, RBF, VM
E2	Call attempt	TL, 3WC
E3	B matches an entry in TCS list	TCS
E4	Flash signal	3WC
E5	Called party answer	RC

7.4.4 Step 4: Linked Events Identification

In this step, the developer identifies linked trigger events. The results of this step in the telephony features case study are presented in Table 7.4.

Table 7.4: Linked events identification table for the telephony features case study

Event ID	Event Description	Linked to	Mathematical Representation
E1	4Call request	E3, E4, E5	E1 ~> E3 E1 ~> E4 E1 ~> E5
E2	Call attempt	E4, E5	E2 ~> E4 E2 ~> E5
E4	Flash signal	E1, E2	E4 ~> E1 E4 ~> E2
E5	Called party answer	E4	E5 ~> E4
Where E_i ~> E_j indicated that event E_j is linked to event E_j			

7.4.5 Step 5: Trigger Events Charts Representation

In this step, graphical trigger events charts are used to group and graphically represent dynamic behaviour features from Table 7.2 that are triggered by the same trigger events. Figure 7.1 shows the trigger events charts obtained in the telephony features case study. Note that the upside-down triangle in E5 is a negation of event E5: If E5 is false and E1 happens then VM will be activated. This is a special case because the VM requires E1 to happen and E5 must not occur.

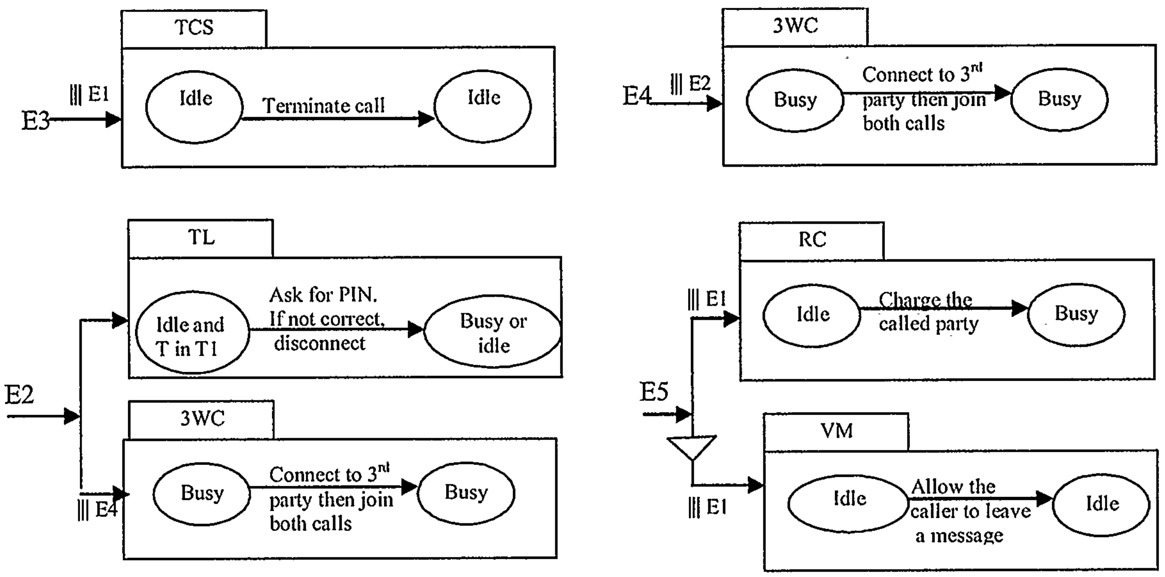


Figure 7.1: Trigger events chart for the telephony features case study

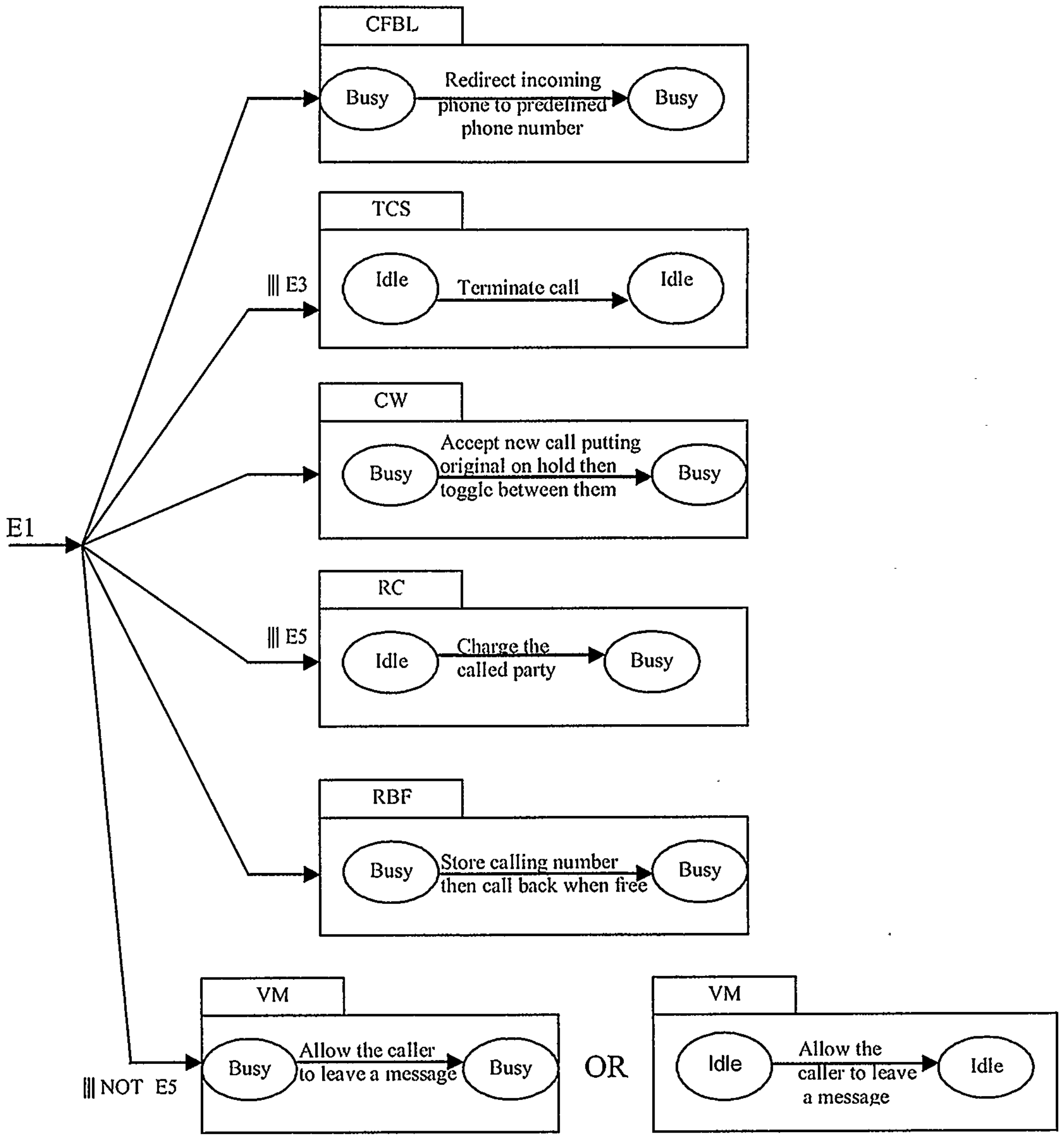

CFBL
Busy
Redirect incoming phone to predefined phone number
Busy
TCS
||| E3
Idle
Terminate call
Idle
CW
Busy
Accept new call putting original on hold then toggle between them
Busy
E1
RC
||| E5
Idle
Charge the called party
Busy
RBF
Busy
Store calling number then call back when free
Busy
VM
Busy
Allow the caller to leave a message
Busy
||| NOT E5
OR
VM
Idle
Allow the caller to leave a message
Idle

Figure 7.1-Continued: Trigger events chart for the telephony features case study

7.4.6 Step 6: Interaction Detection

7.4.6.1 Summary of the Detected Interactions

According to the interaction taxonomy presented in Chapter 3, there are 9 main interaction categories under which all interaction scenarios reside. However, in the telephony features case study, there are no system axioms or resources that have been identified. All the features used in this case study are dynamic behaviour features. Hence, only the main interaction category ③ "Interactions between Two Dynamic Behaviour Features" is relevant in this case study.

Table 7.5 presents a summary of the detected interactions. As can be seen in Table 7.5, IRIS was able to detect 21 interactions among the set of 8 telephony features used in the case study. However, IRIS missed 2 interactions. A discussion about the obtained results is provided in Section 7.5. To provide a better understanding of how these results were obtained, a detailed description is given in the next subsections on the application of the interaction scenarios used for detecting interactions.

Table 7.5: Summary of detected interactions in the telephony features case study

	CFBL	TL	TCS	CW	3WC	RC	RBF	VM
CFBL	▨	✕	SCR15	SCR8	SCR15	SCR15	SCR8	SCR8
TL		▨			SCR15		✕	SCR13
TCS			▨	SCR15	SCR15		SCR15	SCR15
CW				▨	SCR15		SCR8	SCR8
3WC					▨	SCR15	SCR15	SCR15
RC						▨	SCR8	SCR8
RBF							▨	SCR15
VM								▨

Table Symbols:
SCRi: Interaction detected using the *i*th interaction scenario
✕: Missed or wrongly detected interaction

7.4.6.2 Interactions According to Main Interaction Category ③

The interaction category ③, "Interactions between Two Dynamic Behaviour Features", contains interaction scenarios aimed at detecting interactions between two dynamic behaviour features. The basic core of IRIS contains five interaction scenarios to be used under this category: SCR8, SCR10, SCR11, SCR12, and SCR13. However, 4 additional interaction scenarios plug-ins were used which are: SCR9, SCR14, SCR15, and SCR16.

To detect interactions between two dynamic behaviour features, the developer first looks at trigger events charts in Figure 7.1 and the linked events table shown in Table 7.4 and identifies unique pairs of features that are triggered by the same trigger event or by linked trigger events. This resulted in the following pairs of features:

SL(CFBL, TCS), S(CFBL, CW), SL(CFBL, RC), S(CFBL, RBF), S(CFBL, VM), SL(CW, TCS), SDL(RC, TCS), SL(RBF, TCS), SL(VM, TCS), SL(CW, RC), S(CW, RBF), S(CW, VM), SL(RBF, RC), SL(VM, RC), S(RBF, VM), SDL(TL, 3WC), DL(CFBL, 3WC), DL(TCS, 3WC), DL(CW, 3WC), DL(RC, 3WC), DL(RBF, 3WC), DL(VM, 3WC), L(TL, RC),

The notations S(R_i, R_j), L(R_i, R_j), and DL(R_i, R_j) have the same definitions given in Chapter 6. The other notations are defined as follows:

- SL(R_i, R_j): The two requirements R_i and R_j can be triggered by the same trigger event or they can be triggered by linked trigger events such that $E_i \rightsquigarrow E_j$
- SDL(R_i, R_j): The two requirements R_i and R_j can be triggered by the same trigger event or they can be sequentially related through $E_i \rightsquigarrow E_j$ and also they can be sequentially related through $E_j \rightsquigarrow E_i$ (called dual linked events)

Now, the analyst has to analyze the requirements pairs listed above using the 9 interaction scenarios.

A summary of the results of the analysis of the telephony features is listed in Table 7.5. To better understanding these interactions, Table 7.6 presents each interaction along with the scenario that has been used to detect it and an explanation.

Table 7.6: Explanation of telephony features case study interactions

Interaction	Detection Scenario	Explanation
CFBL&TCS	SCR15	A has TCS with B on the screening list. A has CFBL to C. A is busy talking to D. B calls A and hence B is forwarded to C when it should have been screened and rejected. Hence CFBL has bypassed TCS. This is because TCS is activated only when A has idle prestate.
CFBL&CW	SCR8	A has CFBL to B. A has CW. A is busy talking to C. D calls A. The system faces a next state non-determinism situation on which state it should transfer to (CW state or CFBL state).
CFBL&3WC	SCR15	A has 3WC and CFBL to B. A is busy talking to C. A flashes and talks to D then joins both calls with C and D. E Calls A. E gets a busy signal instead of being forwarded to B. This is because A is in a 3WC state which will not allow the activation of CFBL. Hence 3WC has bypassed CFBL.
CFBL&RC	SCR15	A has RC and CFBL to B. A is busy talking to C. D calls A. The system forwards incoming call to another number and A is not charged. Hence CFBL has bypassed RC..
CFBL&RBF	SCR8	A has RBF and CFBL to B. A is busy talking to C. D calls A. The system faces a next state non-determinism situation on which state it should transfer to (RBF state or CFBL state).
CFBL&VM	SCR8	A has VM and CFBL to B. A is busy talking to C. D calls A. The system faces a next state non-determinism situation on which state it should transfer to (VM state or CFBL state).
TL&3WC	SCR15	A has TL and 3WC. B calls A and A answers. A uses the 3WC to place another call to anyone else without having to enter the TL PIN. A was able to do so because TL is activated only when the system has an idle prestate. Hence 3WC has bypassed TL.
TL&VM	SCR13	A has VM and TL. A picks the phone to call VM. A has to enter the TL PIN first. Hence, VM has been negatively impacted by the TL in terms of delay until A enters the required PIN (IF A does not enter the PIN then the TL will override the VM and prevents A from accessing his voice mail)

Table 7.6-Continued: Explanation of telephony features case study interactions

Interaction	Detection Scenario	Explanation
TCS&CW	SCR15	A has CW and TCS with B on the screening list. A is busy talking to C. B calls A and he gets through and is put on hold although he should have been screened. Hence CW has bypassed TCS. This is because TCS works only when A is in an idle prestate.
TCS&3WC	SCR15	A has TCS with B on the screening list. C has 3WC. B calls C. C flashes and uses 3WC to call A and then joins both calls from A and B. B is now talking to A although he should have been screened. Hence 3WC has bypassed TCS.
TCS&RBF	SCR15	A has RBF and TCS with B on the screening list. A receives a call from B and transits to TCS state to initiate a rejection message. At that time, A receives a call from C but RBF is not activated as the system is in TCS state. Hence TCS has bypassed RBF.
TCS&VM	SCR15	A has VM and TCS with B on its screening list. A is busy talking to C. B calls A and VM is activated to allow B to leave a message. Hence VM bypassed TCS because TCS is activated only when the system has an idle prestate.
CW&3WC	SCR15	A has CW and 3WC. A is talking to B and C. D calls A. CW is not activated since the system is in 3WC state. Hence 3WC has bypassed CW.
CW&RBF	SCR8	A has CW and RBF. A is busy talking to B. C calls A. There is a system state non-determinism on which state the system should transfer to (CW or RBF).
CW&VM	SCR8	A has CW and VM. A is busy talking to B. C calls A. There is a system state non-determinism on which state the system should transfer to (CW or VM).
3WC&RC	SCR15	A has 3WC. B has RC. A is busy talking to C then A flashes to use 3WC to call B. A is still being charged for that call although B has RC. This is because the RC works only when the system prestate is in basic call state.
3WC&RBF	SCR15	A has 3WC and RBF. A is busy talking to B and C using 3WC. D calls A. RBF is not activated and D number is not stored because the system is in 3WC prestate. Hence 3WC has bypassed RBF.
3WC&VM	SCR15	A has 3WC and VM. A is busy talking to B. A flashes to make another call to the VM message centre. However, VM is not activated because there is no transition available from a 3WC state to a VM state and hence VM does not work. Hence 3WC has bypassed VM.
RC&RBF	SCR8	A has RC and RBF. A is busy talking to B. C calls A. The system faces a next state non-determinism situation on which state it should transfer to (RC state or RBF state).
RC&VM	SCR8	A has RC and VM. B calls A. The system faces a next state non-determinism situation on which state it should transfer to (RC state or VM state).
RBF&VM	SCR15	A has RBF and VM. A is connected to the message centre to hear his voice mail. This means that the system is in a voice mail state. B calls A. RBF does not start because the system is a VM state not basic call busy state. Hence VM has bypassed RBF.

7.5 Discussion of the Results

7.5.1 Reduction in Number of Comparisons

The developer has to perform 23 comparisons of features using IRIS as explained in Section 7.4.6.2.

When a human expert informally pair wise compares the 8 features used in this case study, s/he would need to carry out 28 comparisons. This means that the application of IRIS resulted in 17.9% fewer comparisons. This percentage cannot be translated to the same percentage of reduction in time and effort, but it still indicates a reduction of time and effort.

7.5.2 Comparing IRIS Results with Other Results Reported in the Literature

Section 7.4 showed the application of IRIS to the telephony features case study. In order to evaluate the efficiency of the results obtained from applying IRIS to the set of 8 telephony features, these results are compared with results obtained by other approaches used by contestants in the second feature interaction contest held in 2000, FIW00 [19]. Table 7.7 shows the results reported by Samborski [142], by Plath and Ryan [152], by Nakamura *et al.* [153] and by Hall [141]. Note that the submission by Plath and Ryan, Samborski, and Hall scores very well whereas the submission by Nakamura comes last.

Table 7.7: Interactions reported by different contestants in the FIW00 contest

	CFBL	TL	TCS	CW	3WC	RC	RBF	VM
CFBL		H P N	H P N	H P N	H P N	H P N	H P S N	H P N
TL				H	H P N S		H P N	H N
TCS				H P N	H P N S		H P N S	P
CW					H P N S		H P N S	H P
3WC						H P N	H P N S	H P N
RC							H P N	P N
RBF								H P N S
VM								

<u>**Table Symbols:**</u>
P: Interaction detected by Plath and Ryan
H: Interaction detected by Hall.
S: Interaction detected by Samborski
N: Interaction detected by Nakamura *et al.*

Table 7.8: Comparing IRIS results to others results from the FIW00 contest

	IRIS	P	N	H	S
Common detected interactions	21	22	21	22	8
Missed interaction	2	1	2	1	15

The outcome of the comparison of IRIS with other results in the literature is shown in Table 7.8. The following explain the results in more details:

1. The row *common detected interactions* in Table 7.8 indicates the number of interactions detected by a specific approach provided that only interactions are counted that were confirmed by at least one other approach.

2. A specific approach is said to have missed an interaction, as indicated in the row *missed interactions* in Table 7.8, if this interaction is detected by at least two other contestants and this specific approach failed to detect it.
3. IRIS is the only approach that uses semi-formal methods. All other approaches reported in Table 7.7 use formal methods.
4. As can be seen from Table 7.8, IRIS missed only 2 interactions which is a very good result considering that it does not use formal methods. The best contestant missed 1 interaction while the worst one missed 15 interactions.
5. As seen from Table 7.5, the two missed interactions had the TL feature as one of the interacting features. The problem with TL is the wait period between the user going off hook and the user entering a valid PIN. It is not clear how to treat this period between going off hook and entering the PIN. This period of time can be treated as a teen line (TL) state or it can be treated as a regular busy state. If this period of time is considered as a TL state then the interaction between CFBL and TL and the interaction between TL and RBF could have been detected using SCR15 as CFBL or RBF would not be triggered because the system is not in a basic call state. However, because we assumed that the system is in a regular basic call state, the two interactions were missed.
6. The interaction between TL and 3WC was detected by IRIS using SCR15 because the bypass would be from 3WC bypassing TL and hence the problem encountered in the point number 5 above does not apply to this interaction. Also, The TL and VM interaction was detected by IRIS because there is an obvious negative impact that can be detected using the interaction scenario SCR13.

7.6 Summary

This chapter presented the application of IRIS to a telephony features case study from the telecommunications domain. The telephony features case study had a set of 8 features that belong to the category of dynamic behaviour features. The application of IRIS resulted in the detection of 21 interactions among the 8 telephony features. IRIS only missed 2 interactions. To evaluate the efficiency of these results, IRIS was compared to the results reported by other approaches in the FIW00 contest and it was able to achieve very good results compared to these formal approaches. Also IRIS achieved a 17.9% reduction in the number of comparisons that a human expert would have to carry out to informally detect interaction among the set of 8 telephony features used in this case study.

CHAPTER EIGHT: APPLYING IRIS IN THE POLICIES DOMAIN - THE SMART HOMES CASE STUDY

8.1 Introduction

This chapter presents the application of IRIS in the policies domain. Hence IRIS is applied to detect interactions at the policies level.

The policy research literature [154-157] has recognized that there are interaction issues between policies and has referred to it as policy conflicts. However, so far very little research has been done to address the problem of policy conflicts. For example, [158] defines policies in a hierarchical way to prevent policy conflicts. However, if a policy in the hierarchy changes, policies can still conflict. The work in [159, 160] promote the use of meta-policies, i.e., policies about creating policies, as a way to prevent conflicts. The work in [161] acknowledges the inevitability of policy conflicts and suggests a negotiation approach for their resolution. The work in [162] describes the use of policies in the telecommunications domain. It suggested the use of a feature interaction manager where policies are used to control the composition of services and features in telephony features, therefore avoiding the problem of feature interactions. The work in [163] proposed a policy architecture for enhancing telephony features and even promoted the use of policies as the features of the future. The work in [68] addresses the problem of interactions in policies but from a social perspective to try to understand what social factors (e.g., stakeholders roles) would cause interactions between two policies.

Most of the work done so far has not comprehensively addressed the problem of policy interactions. For example, the work in [162] and [163] has been limited towards the use of policies in the traditional telecommunications domain. The work in [159] and [161] only look at the prevention of policy interactions and not their detection. However, so far no prevention technique can guarantee that no interactions will occur. Furthermore, so far there has not been a precise definition of when two policies are considered interacting. Even though policies are heavily used in defining user preferences in smart homes, no work has been done so far on investigating policy interactions in this domain.

In this case study, IRIS was applied to detect policy interactions in smart homes among 35 user policies.

This chapter is structured as follows: Section 8.2 presents the concepts of features and policies and a novel view on their relationship especially in smart homes. Section 8.3 presents a description of the smart homes features used in the case study. Section 8.4 shows how IRIS was customized to be applied in the smart homes case study. Section 8.5 shows the application of IRIS to detect interactions as well as the results obtained from each step. Section 8.6 presents a discussion on the obtained results along with a comparison of these results with other results reported in the literature. Finally, Section 8.7 presents a summary of the chapter.

8.2 Features and Policies

During the Feature Interaction Workshop (FIW VII) held in 2003 [30], it became obvious that there is a growing interest in policies and their interactions. However, the differences and interrelationships between policies and features were still very unclear. In this section, a new view on the relationship between features and policies is presented.

8.2.1 Understanding Features and Policies

A *feature* is defined as a coherent and identifiable bundle of system functionality that helps characterize the system from the user perspective [150]. Features are built by system developers as user-requested expansions of a base system. Features have been attractive as they allow the developers of long-lived systems to enrich system performance by adding features over time on top of the base system. An example of a feature in the telecommunications domain is Call Forward on Busy Line (CFBL). CFBL is a feature that, when active, will forward an incoming phone call to a busy subscriber to a predefined phone number.

A *Policy* is defined as information that is used to modify the behaviour of the system [159]. Policies are created by different stakeholders (e.g., normal user, administrator, manager) to reflect personal, organizational or system goals. The attractiveness of policies stems from the fact that people can express their preferences by setting their own policies to customize the system with greater flexibility. An example of a policy set by a user is: "If someone gets out of bed between 10pm and 7am then the lights in the bedroom and the hallway switch on at initially 50% of illumination ramping up to 100% over 1 minute and the bathroom fan is switched on. After leaving the bathroom, the bathroom light and the bathroom fan automatically switch off. After the person gets back into bed the bedroom light is dimmed from 100% to 50% over 1 minute and then switched off'.

There is a major difference between features and policies. The user has very limited control, if any, over the behaviour of a feature. He can activate or deactivate the feature or supply a certain value for a parameter of the feature. But s/he cannot customize the feature to work in a certain way to meet his/her needs. For example, consider the feature Teen Line (TL) from the telecommunications domain which restricts outgoing calls from the phone during a predefined time period unless a PIN is provided. The only control that a user has over this feature is to activate/deactivate it, specify the restricted time frame, and change the PIN. But the user cannot customize this feature to allow outgoing calls in case of emergencies, such as fire, to allow anyone to call 911. However, such customization is possible with policies. For instance, the user can set the policy: "The system shall override the Teen Line PIN restriction when the fire alarm is triggered".

8.2.2 Relationship between Features and Policies

As defined earlier, a feature is a bundle of system functionalities. This means that each feature provides different functionalities to the system. For example, the windows control feature is a feature that allows the control of windows within a smart home and contains the following functionalities:

O_{w1}: The windows can be opened/closed at any time by occupants using a remote control.

O_{w2}: The system shall open/close the windows between time X1 and time X2

O_{w3}: The system shall open/close the windows when day/night begins

where O_{wi}: the operation i associated with feature w (windows)

Now, a policy is information that is used to allow the modification of system behaviour. Policies achieve this modification and customization of system behaviour through the invocation of one or more functionalities within one or more features. For example, consider the following policy set by an occupant in a smart home: "Open the windows between 5:00 pm and 6:30 pm". This policy invokes only one functionality, O_{w2}, in the windows control feature and executes the action of opening the windows when the clock of the system indicates 5:00 pm and closes them again at 6:30 pm.

The relationship between features and policies can be described from an object oriented perspective: A policy is a specific *Run* that the user wants the system to execute to exhibit a specific behaviour using values that accommodate his/her special needs. Now, the system is composed of a set of features. Each feature can be thought of as an *Object.* Also, each feature will have different functionalities in it (e.g., the windows control feature). These functionalities can be considered as *Methods*.

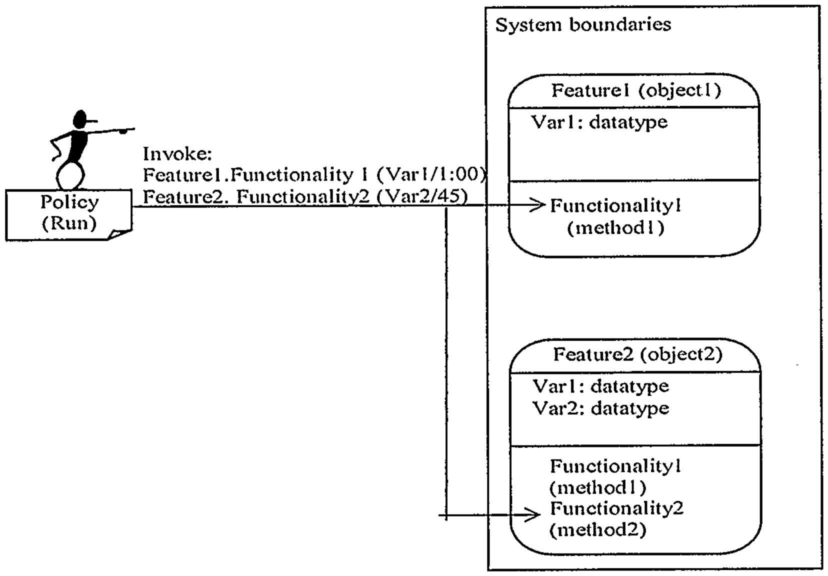


Figure 8.1: Object-oriented description of relationship between features and policies

A user can then set a *run* with specific values that invokes one or more *methods* from the same or different *objects* replacing the parameters in these methods with the values provided by the user in the run. A similar concept describes the relationship between features and policies. The user can set a policy (*a run*) with specific values to replace parameters in the functionalities. This policy will invoke one or more functionalities (*methods*) from the same or different features (*objects*) replacing the parameters within these functionalities with the values provided by the user in the policy to achieve its task. The diagram in Figure 8.1 illustrates this point.

8.2.3 Features and Policies in a Smart Home Architecture

After defining features, policies, and their relationship, we now want to describe how features and policies look like in a smart home architecture. Figure 8.2 presents an overall architecture showing policies, features and physical elements of smart homes. Physical elements are responsible for carrying out the physical actions of the different functionalities when triggered (e.g., actuators, appliances, air conditioning, heating, light bulbs, etc).

The policy layer contains all the policies of the smart home including policies set by the occupants (user policy) or policies that are set by the system administrator and developer (system policy). The feature layer includes all the features within the smart home. Finally, the physical elements layer contains all the physical elements that are connected to the smart home network. Usually, all physical elements are connected to a central network where the master control software coordinates all the operations of the physical elements.

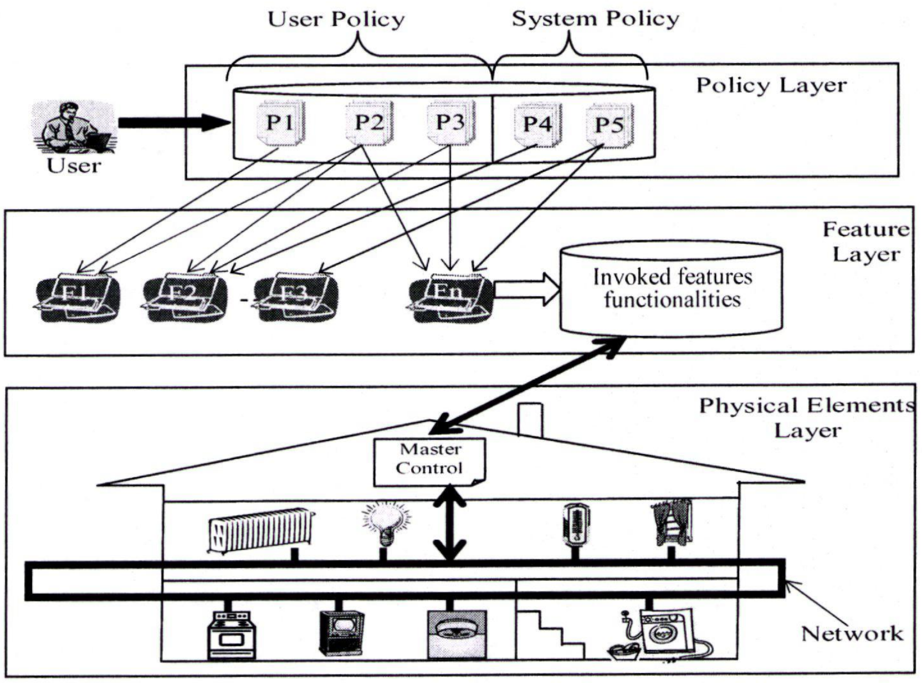


Figure 8.2: Features, policies, and physical elements within smart homes

When a user has a certain preference for the behaviour of one or more physical elements he defines a user policy in the policy layer describing his preference. This user policy then invokes the functionalities controlling the behaviour of the affected physical elements. The invoked functionalities pass on the user preferences described in the user policy to the master control in the physical elements layer. Finally, the master control activates the required physical elements according to the user-defined behaviour.

8.2.4 Simple Policies and Compound Policies

Smart homes are controlled by users setting different policies according to their preferences. However, the complexity of these policies can vary greatly. Therefore, we introduce the concepts of *simple policy* and *compound policy.* A simple policy is a policy that causes a direct invocation of only one functionality in only one feature. A compound policy is a policy that causes an invocation of more than one functionality within the same or different features. In other words, a compound policy can be seen as the concatenation of two or more simple policies. Consider a policy that states "Close the water tap when the water level reaches 75% of the sink in the kitchen". This policy is considered a simple policy because it directly invokes only one functionality which is P11.1 in the Water Overflow Control Feature (see section 8.3). On the other hand, the policy "Close the water tap when the water level reaches 75% of the sink in the kitchen and call the main occupant of the house on his cell phone", is considered a compound policy because it invokes two functionalities, namely: functionality: (1) "Close the water tap when the water level reaches 75% of the sink in the kitchen", and (2) "Call the main occupant of the house on his cell phone" in the communication feature.

According to the above discussion, detecting policy interactions in general can be achieved by detecting simple policy interactions. This even can provide more precise results than detecting interactions between two compound policies. This is because detecting interactions at the simple policy level can detect interactions within a single compound policy by detecting interactions between two simple policies in the body of this compound policy.

Since by definition a simple policy invokes only one functionality in one feature, detecting interactions between functionalities is equivalent to detecting interactions between simple policies. Therefore, the remainder of this chapter uses the terms functionality and simple policy interchangeably.

8.3 Smart Homes Features

Before IRIS can be applied to the smart homes case study, the features of a smart home have to be defined. Smart homes can contain many different features, some of which are not very common due to their high cost and technical difficulties. Furthermore, many features are designed to help people with specific disabilities and therefore are not installed in all smart homes. This case study investigates only the common features that are likely used in most smart homes. Figure 8.3 shows an overview of these features.

Feature 1: Intruder Alarm Feature

This is a security feature. The occupants can activate/deactivate the intruder alarm from inside the house using the alarm switch. The intruder alarm feature, when active, can be triggered by a magnetic reeds sensor indicating that a window has been opened, by the main door lock sensor indicating that the main door lock has been opened, by a Passive Infra Red (PIR) sensor indicating movement in some areas, or by pressure pads indicating that a person stepped on a predefined area.

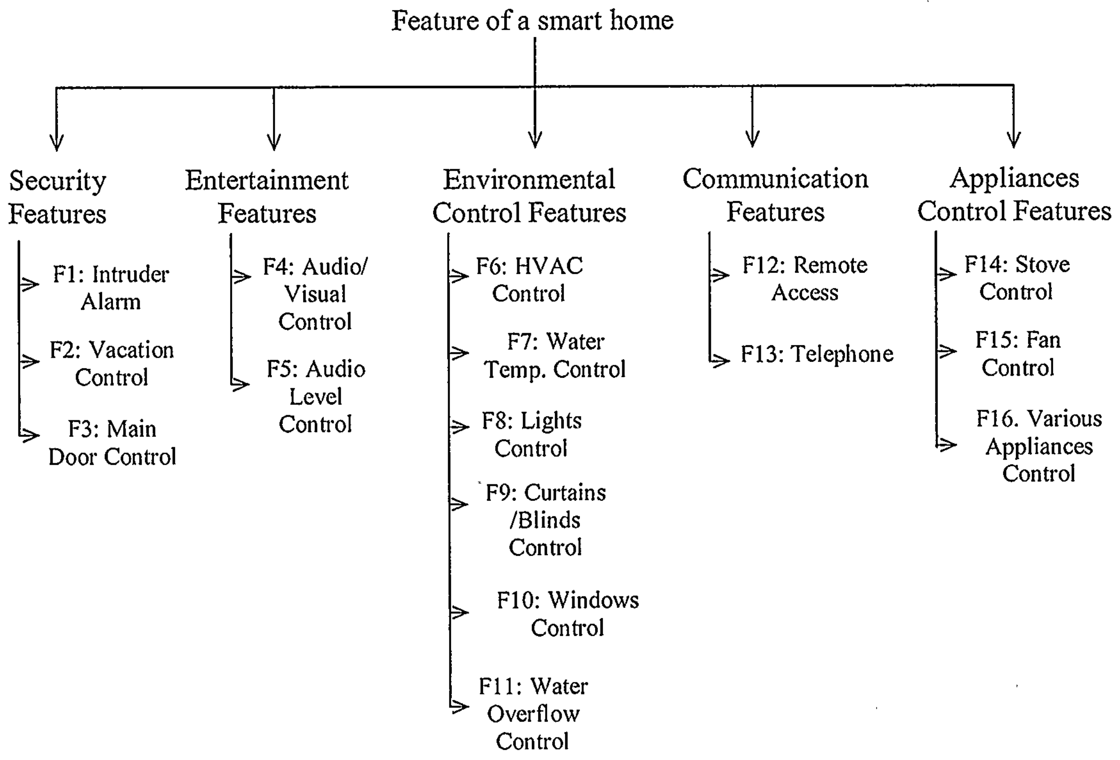


Figure 8.3: Overview of the features of a smart home

Feature 2: Vacation Control Feature

This feature can be used when the occupants are on vacation for an extended period of time. It uses predefined time settings to automatically turn on TV and lights for 60 minutes in predefined areas. The feature is activated/deactivated by a switch from the interior of the house.

Feature 3: Main Door Control Feature

This feature that locks the main door lock of the house using an electronic lock when the main door is shut. The occupants can use an interior switch to unlock and open the main door from the inside. For safety purposes the main door automatically unlocks and opens when the Gas/Heat/Smoke sensor is triggered.

Feature 4: Audio/Visual Control Feature

This feature allows the occupants to control A/V devices through remote controls or to ask the system to turn certain A/V devices on/off at predefined time settings.

Feature 5: Audio Level Control Feature

This feature allows the occupants to preset the audio level of different A/V devices to certain levels when they are turned on during the day or night. It also allows the occupants to set a maximum audio level throughout the house that cannot be exceeded. This maximum audio level is chosen by the occupant to avoid loud noise or disturbance during the day/night.

Feature 6: Heating, Ventilation and Air Conditioning Control Feature

The Heating, Ventilation and Air Conditioning (HVAC) control feature controls the temperature of the house. This feature increases/decreases the temperature inside the house to a user-preset temperature when the thermostats' readings are different from this preset temperature. This feature also allows the occupants to define a program to increase/decrease the temperature of the house at predefined time intervals.

Feature 7: Water Temperature Control Feature

This feature controls the temperature of the hot water in the house. It maintains the temperature of the hot water from the hot water tap in the kitchen at 45 ^{o}C and that of the hot water tap in the bathroom at a temperature of 40 ^{o}C

Feature 8: Lights Control Feature

This feature controls the intensity of light inside the house. It increases/decreases light intensity to correspond to the increase/decrease of a light dimmer. During the night, this feature increases the light intensity in a certain part of the house to the maximum within 2 minutes when a positive PIR signal is received from that part. When the PIR signal is negative for 15 minutes, the lights are automatically switched off. Finally, this feature can be set to automatically turn on the lights according to a daylight sensor when the night begins.

Feature 9: Curtains and Blinds Control Feature

This feature can be used to automatically open/close the curtains and blinds in a certain area at predetermined time settings. It can also be set to automatically open/close the curtains and blinds in a certain area according to a daylight sensor.

Feature 10: Windows Control Feature

This feature opens/closes the windows in predefined areas based on predefined time settings.

Feature 11: Water Overflow Control Feature

This safety feature shuts down the water tap when the water reaches or exceeds 75% of the total volume of the sink in the kitchen or the tub in the bathroom.

Feature 12: Remote Access Feature

This feature allows the occupants to remotely activate any feature within the smart home from any location via the telephone. The occupants call the home phone number, and when there is no answer after a user-defined number of rings a remote access module is activated asking for a PIN to allow the remote control of home features.

Feature 13: Telephone Feature

This feature enforces the presence of a Plain Old Telephone Service (POTS) [151] or Voice over Internet Protocol (VoIP) telephone line [164]. It has an answer machine installed to record messages when receiving a phone call with no answer for a certain number of rings.

Feature 14: Stove Control Feature

This safety feature can be used to shut down and prevent any activation of the stove during predefined time periods. This feature is also used to shut down the stove when the Gas/Heat/Smoke sensor is triggered.

Feature 15: Fan Control Feature

This feature automatically turns on the kitchen fan when the humidity sensor is triggered. When the sensor signal is lost for 20 minutes while the fan is on, , the fan is automatically switched off.

Feature 16: Control of Various Appliances Feature

This feature allows occupants of the house to control various appliances like the food processor, water boiler, etc. using remote controls.

8.4 Customizing IRIS for the Smart Homes Case Study

8.4.1 Plug-ins used in the Smart Homes Case Study

The domain of smart homes is relatively new. It contains numerous features 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. Since each feature in the smart homes is complex and describes many functionalities in its body, the plug-in "Functionalities Identification" was used to break down the complex textual description of each feature into atomic simple functionalities. This plug-in is inserted into hook H1 as a complete step prior to IRIS step 1.
2. Several functionalities in the case study had many parameterized parts in their textual descriptions. Hence, the plug-in "Parameters Assignment" was used to replace these parameterized textual parts with parameters such as X or Y. This plug-in is inserted also into hook H1 prior to IRIS basic core step 1.

3. The parameters identified from the execution of the plug-in “Parameters Assignment” means that the textual requirements have parameters in their body and they must be dealt with using the appropriate attributes. In order to do this, the plug-in “Parameters” was used to enforce the use of the attribute “Parameters”. Recall that the attribute Parameters was an optional attribute and is not used unless there are parameters in the textual description of requirements. This plug-in is applied during step 2 of the basic core of IRIS.
4. The plug-in “Parameters Range” must be used in order to indicate the allowed range of values that each parameter can have. The values assigned to each parameter have a major influence on possible interactions between functionalities. This plug-in is applied during IRIS basic core step 2.
5. To detect all possible interactions between system axioms simple policies and dynamic behaviour simple policies, the plug-ins interaction scenarios “SCR5”, “SCR6”, and “SCR7” were inserted into hook H8. These interaction scenarios are applied as part of the sixth step of IRIS.
6. To detect all possible interactions between dynamic behaviour simple policies, four plug-ins interaction scenarios have been inserted as plug-ins into hook H8, namely: SCR9, SCR14, SCR15, and SCR16. These interaction scenarios are applied as part of the sixth step of IRIS.

8.4.2 Assumptions used in the Smart Homes Case Study

In the case study described in this chapter, the following assumptions were made:

1. IRIS is being applied to detect interactions at the policy level.
2. Interaction detection between functionalities is equivalent to interaction detection between user policies (see section 8.2.4).
3. Throughout the case study the two terms simple policy and functionality are equivalent and are used interchangeably (see section 8.2.4)
4. The vacation control feature is assumed to turn on/off TV and lights at predefined time settings. This limitation was imposed for simplicity.
5. Only the answer machine feature from the set of traditional telecommunications features (Feature 13) was used because answer machine can be installed without having to install a more comprehensive set of features.
6. The features used in the case study were defined by the investigator based on several different resources (e.g., [165-168]) as no complete definitions for the smart homes features were found in one resource.
7. All devices and sensors are connected to a central network controlled by the master control software. This master control software is used to control and coordinate all operations of the different devices based on user policies.
8. States are described using state variables. Within a certain state only variables of interest to the policy under investigation are listed. This simplification is possible since other state variables have no effect on the outcome of the interaction detection step.

8.5 Applying IRIS to Detect Interactions in the Smart Homes Case Study

8.5.1 Using Plug-ins "Functionalities Identification" and "Parameters Assignment"

As mentioned in Section 8.4.1, the two plug-ins "Functionalities Identification" and "Parameters Assignment" are applied at the beginning prior to the execution of the first step of IRIS basic core. The application of the two plug-ins are presented together in this subsection. It must be noted that the two plug-ins are independent and can be executed in any order, i.e., it is possible to identify the functionalities within each feature first then look for parameterized text and assign it to parameters. Also, it is possible to identify parameterized text within features first and assign it to parameters then identify different functionalities within each feature.

Each functionality (simple policy) is given a unique ID that starts with a P followed by the number of the feature and the number of the functionality/simple policy (e.g. P3.2 stands for simple policy number 2 in feature 3).

- **Functionalities (simple policies) in the Intruder Alarm Feature**

P1.1: Activated/deactivated by a switch from inside the house called alarm switch.

P1.2: Alarm is triggered when the feature is active and a magnetic reed sensor indicates that a window is being opened

P1.3 Alarm is triggered when the feature is active and the main door lock sensor indicates that the main door lock is being opened

P1.4 Alarm is triggered when the feature is active and a PIR sensor indicates movement in X1, where X1: Location, X1= {Living room, Bedrooms, Hallway, Kitchen}

P1.5 Alarm is triggered when the feature is active and pressure pads indicate the presence of a person in X2, where X2: location, X2= {Living room, Bedrooms, Hallway}

- **Functionalities (simple policies) in the Vacation Control Feature**

P2.1 Activated/deactivated by a switch from inside the house called vacation switch.

P2.2 Turns on TV for 60 minutes at X3, where X3: Time, X3=00:00-23:59

P2.3 Turns on lights for 60 minutes at X4 in X5, where X4: Time, X4=00:00-23:59 and X5: Location, X5= {Living room, Bedrooms}

- **Functionalities (simple policies) in the Main Door Feature**

P3.1 Locks the main door lock of the house when the main door is shut.

P3.2 Occupants can unlock and open the main door from inside by interior switch

P3.3 Unlocks and opens the main door when the Gas/Heat/Smoke sensor is triggered.

- **Functionalities (simple policies) in the Audio/Visual Control Feature**

P4.1 Occupants can control all A/V devices through remote controls

P4.2 Turns on/off X6 A/V device at X7, where X6: A/V device, X6={TV, CD, DVD} and X7: Time, X7=00:00-23:59

- **Functionalities (simple policies) in the Audio Level Control Feature**

P5.1 Presets the audio level of audio device X8 to X9 when turned on, where X8: A/V device, X8={TV, CD, DVD} and X9: Audio level, X9 = {1..63}

P5.2 Occupants can set X10 as a maximum audio level throughout the house, where X10: Audio level, X10 = {1..63}

- **Functionalities (simple policies) in the Heating, Ventilation and Air Conditioning Control Feature**

P6.1 Increases/decreases the ambient temperature inside the house to X11 when the readings from the thermostats are different from this preset temperature, where X11: Temperature, X11 = {15..35}

P6.2 Increases/decreases the temperature of the house to X12 at X13, where X12: Temperature, X12 = {15..35} and X13: Time, X13=00:00-23:59

- **Functionalities (simple policies) in the Water Temperature Control Feature**

P7.1 Maintains the temperature of the hot water from the hot water tap in the kitchen at 45 degree centigrade.

P7.2 Maintains the temperature of the hot water from the hot water tap of the bathroom at 40 degree centigrade.

- **Functionalities (simple policies) in the Lights Control Feature**

P8.1 Increases/decreases the light intensity to correspond to the increase/decrease of a light dimmer .

P8.2 Increases the light intensity during night in X14 to the maximum within 2 minutes when a positive PIR signal is received from X14, where X14: Location, X14= {Living room, Bedrooms, Bathroom}

P8.3 Automatically shuts down the lights during night in X15 when a PIR signal is negative for 15 minutes from X15, where X15: Location, X15= {Living room, Bedrooms, Bathroom, Hallway}

P8.4 Automatically turns on the lights according to a daylight sensor when the night begins.

• **Functionalities (simple policies) in the Curtains and Blinds Control Feature**

P9.1 Automatically opens/closes the curtains and blinds in X16 at X17, where X16: Location, X16= {Living room, Bedroom} and X17: Time, X17=00:00-23:59

P9.2 Automatically opens/closes the curtains/blinds in X18 according to daylight sensor, where X18: Location, X18= {Living room, Bedroom}

• **Functionalities (simple policies) in the Windows Control Feature**

P10.1 Opens/closes the windows in X19 at X20, where X19: Location, X19= {Living room, Bedroom} and X20: Time, X20=00:00-23:59

• **Functionalities (simple policies) in the Water Overflow Control Feature**

P11.1 Closes the water tap when the water reaches or exceeds 75% of the total volume of the sink or the tub either in the kitchen or in the bathroom

• **Functionalities (simple policies) in the Remote Access Feature**

P12.1 Activates a remote access module when an incoming telephone call has not been answered within X21 rings, where X21: number of phone rings, X21 = {2..8}

• **Functionalities (simple policies) in the Telephone Feature**

P13.1 Enforces the presence of a telephone line with either standard POTS or VOIP

P13.2 Activates an answer machine to record messages when receiving a call with no answer for X22 rings, where X22: number of phone rings, X22 = {2..8}

• **Functionalities (simple policies) in the Stove Control Feature**

P14.1 Shut down and prevent any activation of the stove during X23 and X24, where X23 and X24: Time, X23 and X24=00:00-23:59

• **Functionalities (simple policies) in the Fan Control Feature**

P15.1 Automatically turns on the kitchen fan when the humidity sensor is triggered

P15.2 Automatically switches off the kitchen fan when the humidity signal is lost for 20 minutes while the fan is on

• **Functionalities (simple policies) in the Control of Various Appliances Feature**

P16.1 Occupants can control various appliances like the food processor, water boiler…etc. using remote controls

8.5.2 Step 1: Simple Policies Classification

The first step is used to organize the simple policies into system axiom simple policies and dynamic behaviour simple policies. The results of the application of the first step are shown in Table 8.1.

Table 8.1: Classification table for the smart homes case study

System Axioms Simple policies	P4.1, P5.2, P7.1, P7.2, P13.1, P16.1
Dynamic Behaviour Simple Policies	P1.1, P1.2, P1.3, P1.4, P1.5, P2.1, P2.2, P2.3, P3.1, P3.2, P3.3, P4.2, P5.1, P6.1, P6.2, P8.1, P8.2, P8.3, P8.4, P9.1, P9.2, P10.1, P11.1, P12.1, P13.2, P14.1, P14.2, P15.1, P15.2.

8.5.3 Step 2: Simple Policies Attributes Identification

This step identifies different attributes within the smart homes policies. Table 8.2 contains attributes of system axioms whereas Table 8.3 contains attributes for dynamic behaviour simple policies. It is worth saying that the execution of the two plug-ins "Parameters" and "Parameters Range" resulted in adding two columns to Tables 8.2 and 8.3.

State variables were used to describe the attributes pre-state and next state of the system in Table 8.3. For example, MainDoorLock=closed states that the main door lock is in a closed state. Not every state will have a value assigned to it. The value DxR in the table corresponds to a "don't care" value. The don't care value is necessary to represent certain cases as in P6.2 where it does not matter what the value of the previous temperature is because the objective of P6.2 is to increase/decrease the temperature to the predefined setting regardless of the previous temperature.

Table 8.2: System axioms attributes identification table for the smart homes policies

ID	Description	Rule	Condition	Parameters	Parameters Range
P4.1	Occupants can control all A/V devices through remote controls	Control all A/V devices through remote controls	True	-	-
P5.2	Occupants can set X10 as a maximum audio level throughout the house	Set X10 as a maximum audio level throughout the house	True	X10:Audio level	{1..63}
P7.1	Maintains the temperature of the hot water from the hot water tap in the kitchen to 45 oC	Maintains the temperature of the hot water of the hot water tap in the kitchen to 45 oC	True	-	-
P7.2	Maintains the temperature of the hot water from the hot water tap of the bathroom to 40 oC.	Maintains the temperature of the hot water of the hot water tap of the bathroom to 40 oC.	True	-	-
P13.1	Enforces the presence of a telephone line with either standard POTS or VOIP	A telephone line is always present with either standard POTS or VOIP.	True	-	-
P16.1	occupants can control various appliances like the food processor, water boiler…etc by remote controls	Control various appliances by remote control	True	-	-

Table 8.3: Dynamic behaviour attributes identification table for the policies

ID	Description	Pre-state	Trigger Event	Action	Next State	Parameters	Parameters Range
P1.1	Security Alarm is Activated/deactivated by a switch from inside the house called alarm switch.	SecurityAlarm= off/on	Activate/ deactivate security alarm switch is pressed	Activate/ deactivate security alarm	SecurityAlarm = on/off	-	-
P1.2	Alarm is triggered when the feature is active and a magnetic reeds sensor indicates that a window is being opened	SecurityAlarm = on, Alarm=not_set, Windows(DxR) =closed	Window is opened	Set security alarm	SecurityAlarm =on, Alarm=set, windows(DXR)= open	-	-
P1.3	Alarm is triggered when the feature is active and the main door lock sensor indicates that the main door lock is being opened	SecurityAlarm = on, Alarm=not_set, MainDoorLock =closed	Main door lock is opened	Set security alarm	SecurityAlarm =on, Alarm=set, MainDoorLock= open	-	-
P1.4	Alarm is triggered when the feature is active and a PIR sensor Indicates movement in X1	SecurityAlarm = on, Alarm=not_set, PIR=negative	Movements in X1	Set security alarm	SecurityAlarm =on, Alarm=set, PIR= positive	X1: Location	{LivRm, BdRm, Hall, kitch}
P1.5	Alarm is triggered when the feature is active and pressure pads indicate the presence of person in X2.	SecurityAlarm = on, Alarm=not_set, PressurePad= negative	pressure pad in X2 is pressed	Set security alarm	SecurityAlarm =on, Alarm=set, PressurePad= positive	X2: location	{LivRm, BdRm, Hall}
P2.1	Vacation Control is Activated/deactivated by a switch from inside the house called vacation switch	VacationControl =off/on	Activate/ deactivate vacation control switch Is pressed	Activate/ deactivate vacation control	VacationControl = on/off	-	-
P2.2	Vacation control Turns on TV for 60 min. at X3	VacationControl =on, TV=off	Time=X3	Turn on TV for 60 min.	VacationControl =on, TV=on	X3: Time	{00:00-23:59}
P2.3	Vacation control turns on lights for 60 minutes at X4 in X5	VacationControl =on, Lights(X5)=off	Time=X4	Turn on lights for 60 min. in X5	VacationControl =on, Lights(X5)=on	X4: Time, X5: Location	X4={00:00-23:59} X5= LivRm, BdRm}
P3.1	Main Door lock feature will Lock the main door lock of the house when main door shut.	MainDoor=open MainDoorLock =open	Main door is shut	Lock the main door lock	MainDoor =closed MainDoorLock =closed	-	–
P3.2	Occupants can unlock and open the main door from inside by interior switch	MainDoor =closed MainDoorLock =closed	Unlock main door switch is pressed	Unlock the main door lock and open the main door	MainDoor= open MainDoorLock =open	-	-
P3.3	Unlocks and opens the main door when the Gas/ Heat/ Smoke sensor triggers.	MainDoor =closed MainDoorLock =closed	Gas/heat/ Smoke sensor is triggered	Unlock the main door lock and open the main door	MainDoor= open MainDoorLock =open	-	-
P4.2	Turns on/off X6 A/V device at X7	X6On=false/true	Time=X7	Turn on/off the A/V device X6	X6On=True/false	X6: A/V device, X7: Time,	X6={TV, CD, DVD} X7={00:00-23:59}
P5.1	Presets the audio level of audio device X8 to X9 when turned on	X8On=False X8Audio_ level =DxR	X8 is turned on	Preset the audio level of X8 to X9	X8On=True, X8Audio_level= X10	X8:A/V device, X9: level	X8={TV, CD, DVD}, X9={1..63}
P6.1	Increases/Decreases the temp. inside the house to X11 when the reading from thermostats are different from this preset temp.	Temp=DxR	Thermostats ≠ X11	Increases/ Decreases the temp. inside the house to X11	Temp=X11	X11: Temp.	{15..35}
P6.2	Increases/decreases the temperature of the house to X12 at X13.	Temp=DxR	Time=X13	Increase/ decrease temp of the house to X12	Temp=X12	X12: Temp. X13: Time	X12 = {15..35} X13={00:00-23:59}

Table 8.3 –Continued: Dynamic behaviour attributes identification table

ID	Description	Pre-state	Trigger Event	Action	Next State	Parameters	Parameters Range
P8.1	increases/decreases light intensity to correspond to the increase/decrease of a light dimmer slider	LightIntens =DxR	Increase/ decrease of the dimmer slider	Increase/ decrease light intensity to match increase/ decrease of the slider	LightIntens. =DimmmerSlider	-	-
P8.2	Increases the light intensity during night in X14 to a maximum within 2 minutes when a positive PIR signal is received from X14.	Daylight=false Lights(X14)=off LightIntens(X14)=0	Movement in X14	increase the light intensity in X14 to a max. within 2 minutes	Daylight=false Lights(X14)=on LightIntens.(X14) =max	X14: Location	{LivRm, BdRm, bathRm}
P8.3	Automatically shuts down the lights during night in X15 when a PIR signal is negative for 15 minutes from X15.	Daylight=false Lights(X15)=on LightIntens (X15)=DxR	No movements in X15 for 15 minutes	Shut down the lights	Daylight=false Lights(X15)=off LightIntens.(X15) =0	X15: Location	{LivRm, BdRm, bathRm, hall}
P8.4	Automatically turns on the lights according to a daylight sensor when the night begins.	Daylight=True Lights=off	Night begins	Turn lights on automatically	Daylight=false Lights=on	-	-
P9.1	Automatically opens/ closes the curtains and blinds in X16 at X17.	CurtainsBlinds (X16)=close /open	Time=X17	Open/close the blinds and curtains in X16	CurtainsBlinds (x16) =open/ close	X16: Location, X17: Time	X16={LivRm, BdRm} X17={00:00-23:59}
P9.2	Automatically open/close the curtains and blinds in X18 according to daylight sensor	Daylight= false /true,CurtainsBlinds(X18)=close /open	Day/night begins	Open/close curtains and blinds in X18	Daylight= true / false, CurtainsBlinds (x18) =open/ close	X18: Location	{LivRm, BdRm}
P10.1	Opens/closes the windows in X19 at X20	Windows(X19) =close/open	Time=X20	Open/close windows in X19	Windows(X19)= open/close	X19: Location, X20: Time	X19={LivRm, BdRm} X20={00:00-23:59}
P11.1	Shuts down the water tap when the water reaches or exceeds 75% of the total size of the sink or the tub either in the kitchen or the bathroom	Tap/shower-Valve=open	Water level ≥75%	Shutdown water	Tap/showerValve= closed	-	-
P12.1	Activates a remote access module when receives a telephone call for X21 rings with no answer	Telephone=idle RemoteAccess= idle	Receive a call request AND no answer for rings =X21.	Activate remote access module	Telephone=busy, RemoteAccess= Active	X21: number of phone rings	{2..8}
P13.2	Activates an answer machine to record messages when receiving a call with no answer for X22 rings	Telephone=idle AnswerMachine =idle	Receive a call request AND no answer for rings=X22	Activate answer machine	Telephone=busy, AnswerMachine= on	X22: number of phone rings	{2..8}
P14.1	Shut down and prevent any activation of the stove during X23 and X24.	Stove=DxR	Time=X23	Shutdown and prevent any activation of the stove till X24	Stove=off	X23, X24 : Time	{00:00-23:59}
P14.2	Shutdown the stove when the Gas/Heat/Smoke sensor is triggered	Stove=DxR	Gas/heat/ Smoke sensor is triggered	Shutdown the stove	Stove=off	-	-
P15.1	Automatically turns on the kitchen fan when the humidity sensor is triggered	KitchenFan=off	Humidity sensor is triggered	Turn on kitchen fan	KitchenFan=on	-	-
P15.2	Automatically shutoff the kitchen fan when the humidity signal is lost for 20 min. while fan is on	KitchenFan=on	Humidity sensor is negative for 20 minutes	Turn off the kitchen fan	KitchenFan=off	-	-

8.5.4 Step 3: Trigger Events Extraction

In this step, the developer identifies and extracts all the different trigger events from Table 8.3. The idea behind this step is to identify different simple policies that are triggered by the same trigger event. The results of this step are shown in Table 8.4.

Table 8.4: Trigger events extraction table for the smart homes case study

Event ID	Event Description	Simple Policies Triggered by this Event
E1	Activate/ deactivate security alarm switch is pressed	P1.1
E2	A Window is opened	P1.2
E3	Main door lock is opened	P1.3
E4	Movements	P1.4, P8.2
E5	Pressure pad is pressed	P1.5
E6	Activate/ deactivate vacation control switch is pressed	P2.1
E7	Time	P2.2, P2.3, P4.2, P6.2, P9.1, P10.1, P14.1
E8	Main door is shut	P3.1
E9	Unlock main door switch is pressed	P3.2
E10	Gas/heat/Smoke sensor is triggered	P3.3, P14.2
E11	A/V device is turned on	P5.1
E12	Thermostats $\neq$ preset temperature	P6.1
E13	Increase/ decrease of the dimmer slider	P8.1
E14	No movements for 15 minutes	P8.3
E15	Day begins	P8.4, P9.2
E16	Night begins	P9.2
E17	Water level $\geq$75%	P11.1
E18	Receive a call request, and no answer	P12.1, P13.2
E19	Humidity sensor is triggered	P15.1
E20	Humidity sensor is negative for 20 minutes	P15.2

8.5.5 Step 4: Linked Events Identification

As explained in Chapter 4, this step is important to identify linked trigger events and hence examine the actions of the policies that might be triggered sequentially by linked events. The results of this step are shown in Table 8.5.

Table 8.5: Linked events identification table for the smart homes case study

ID	Event Description	Linked to	Mathematical Representation
E1	Activate/ deactivate security alarm switch is pressed	E2, E3, E4, E5, E6, E8, E9, E13, E18	E1<~>E2, E1<~>E3, E1<~>E4, E1<~>E5, E1<~>E6, E1 <~> E8, E1~>E9, E1<~>E13, E1~>E18
E2	A Window is opened	E9, E12, E18, E20	E2<~>E9, E2~>E12, E2~>E18, E2~>E20
E3	Main door lock is opened	E2, E4, E5, E6, E11, E13, E14, E18, E20	E3~>E2, E3<~>E4, E3<~>E5, E3<~>E6, E3~>E11, E3<~>E13, E3~>E14, E3~>E18, E3~>E20
E4	Movements	E2, E5, E6, E8, E9, E11, E13, E18	E4~>E2, E4<~>E5, E4~>E6, E4~>E8, E4~>E9, E4~>E11, E4~>E13, E4~>E18
E5	Pressure pad is pressed	E2, E6, E8, E9, E11, E13, E18	E5~>E2, E5~>E6, E5<~>E8, E5~>E9, E5~>E11, E5~>E13, E5~>E18
E6	Activate / deactivate vacation control switch is pressed	E8, E9, E13, E14, E18	E6<~>E8, E6~>E9, E6~>E13, E6~>E14, E6~>E18
E7	Time	Ei , where i=1..20	E7~>Ei , where i=1..20
E8	Main door is shut	E2, E9, E10, E11, E13, E14, E18	E8~>E2, E8~>E9, E8~>E10, E8~>E11, E8<~>E13, E8~>E14, E8~>E18
E9	Unlock main door switch is pressed	E3, E12, E13, E14, E18	E9~>E3, E9~>E12, E9~>E13, E9~>E14, E9~>E18
E10	Gas/heat/Smoke sensor is triggered	E2, E3, E4, E5, E9, E12, E18, E19	E10~>E2, E10~>E3, E10~>E4, E10~>E5, E10~>E9, E10<~>E12, E10~>E18, E10~>E19
E11	A/V device is turned on	E18	E11~>E18
E12	Thermostats ≠ preset temperature	E18, E19, E20	E12~>E18, E12~>E19, E12~>E20
E13	Increase/ decrease of the dimmer slider	E18	E13~>E18
E14	No movements for 15 minutes	E18	E14~>E18
E15	Day begins	E1, E2, E3, E4, E5, E9, E11, W12, E13, E18	E15~>E1, E15~>E2, E15~>E3, E15~>E4, E15~>E5, E15~>E9, E15~>E11, E15~>E12, E15~>E13, E15~>E18
E16	Night begins	E1, E2, E3, E4, E5, E9, E11, E12, E13, E14, E18	E16~>E1, E16~>E2, E16~>E3, E16~>E4, E16~>E5, E16~>E9, E16~>E11, E16~>E12, E16~>E13, E16~>E14, E16~>E18
E17	Water level =75%	E4, E5, E10, E18, E19	E17~>E4, E17~>E5, E17~>E10, E17~>E18, E17~>E19
E19	Humidity sensor is triggere	E18	E19~>E18
E20	Humidity sensor is negative for 20 minutes	E18	E20~>E18

8.5.6 Step 5: Trigger Events Charts Representation

The graphical representation by the trigger events charts facilitates the detection of interactions between the dynamic behaviour simple policies. The result of this step is shown in Figure 8.4.

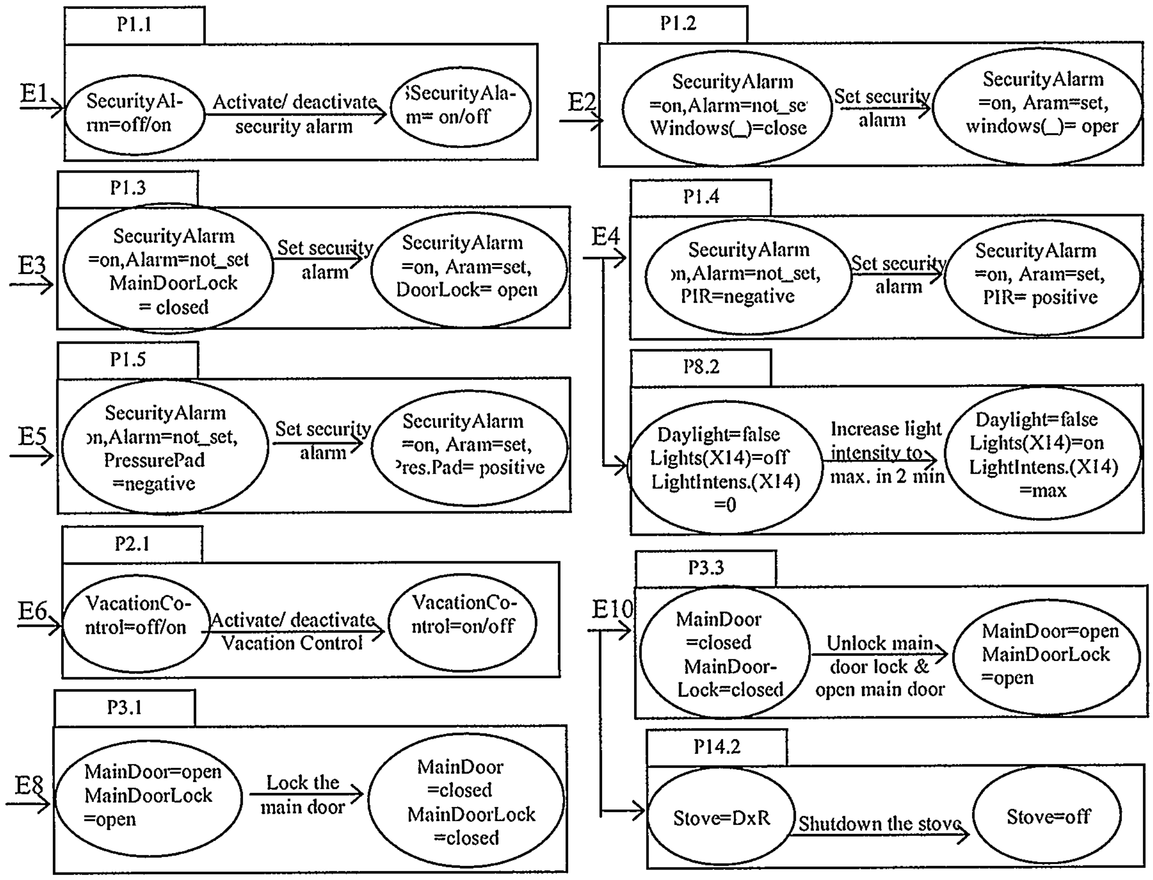
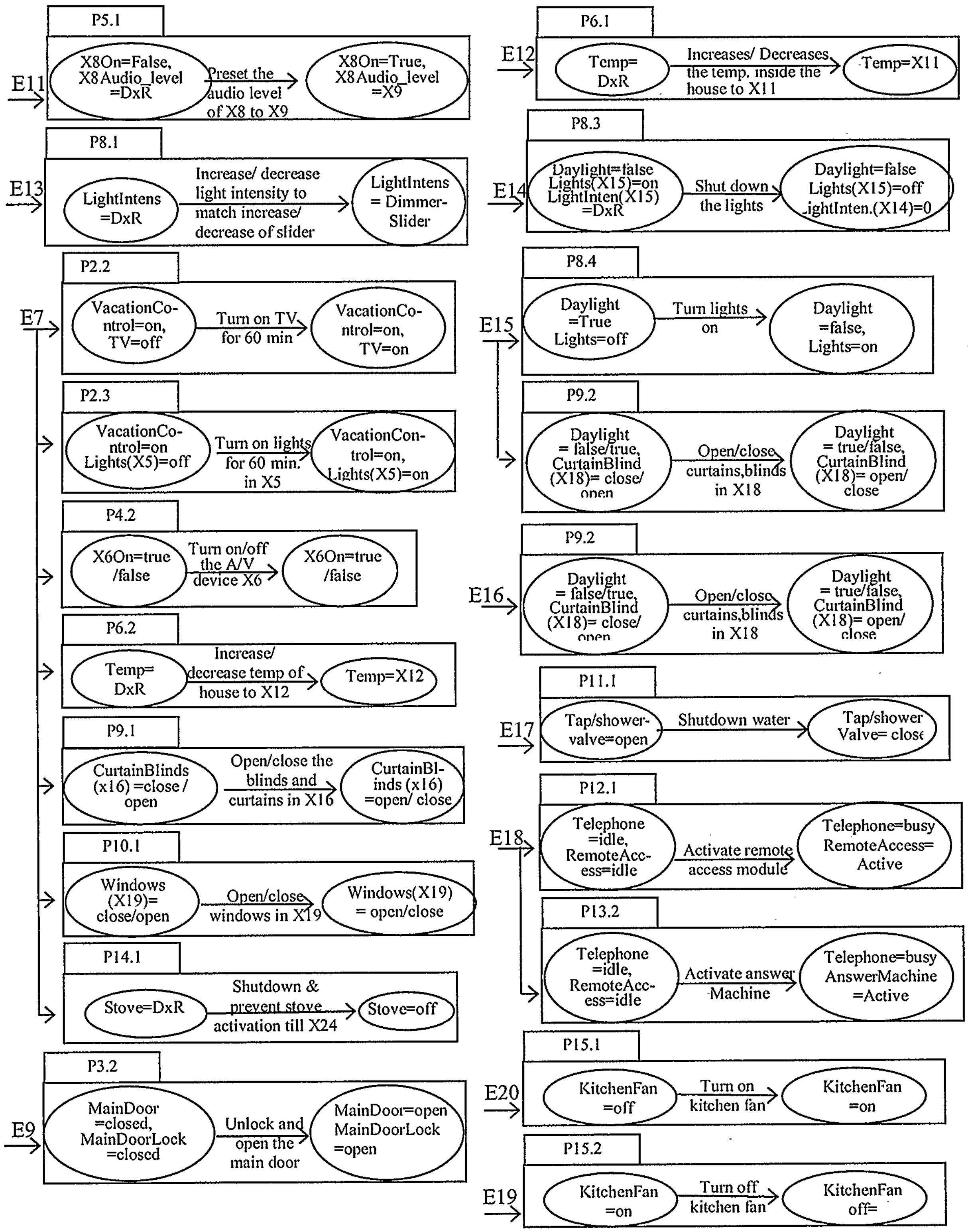

P1.1
E1
SecurityAl-rm=off/on
Activate/ deactivate security alarm
SecurityAla-m= on/off
P1.2
E2
SecurityAlarm =on,Alarm=not_se Windows(_)=close
Set security alarm
SecurityAlarm =on, Aram=set, windows(_)= oper
P1.3
E3
SecurityAlarm =on,Alarm=not_set MainDoorLock = closed
Set security alarm
SecurityAlarm =on, Aram=set, DoorLock= open
P1.4
E4
SecurityAlarm on,Alarm=not_set, PIR=negative
Set security alarm
SecurityAlarm =on, Aram=set, PIR= positive
P1.5
E5
SecurityAlarm on,Alarm=not_set, PressurePad =negative
Set security alarm
SecurityAlarm =on, Aram=set, Pres.Pad= positive
P8.2
Daylight=false Lights(X14)=off LightIntens.(X14) =0
Increase light intensity to max. in 2 min
Daylight=false Lights(X14)=on LightIntens.(X14) =max
P2.1
E6
VacationCo-ntrol=off/on
Activate/ deactivate Vacation Control
VacationCo-ntrol=on/off
P3.3
E10
MainDoor =closed MainDoor-Lock=closed
Unlock main door lock & open main door
MainDoor=open MainDoorLock =open
P3.1
E8
MainDoor=open MainDoorLock =open
Lock the main door
MainDoor =closed MainDoorLock =closed
P14.2
Stove=DxR
Shutdown the stove
Stove=off

Figure 8.4: Trigger events chart of the smart homes dynamic behaviour policies



P5.1
E11
X8On=False, X8Audio_level =DxR
Preset the audio level of X8 to X9
X8On=True, X8Audio_level =X9
P6.1
E12
Temp= DxR
Increases/ Decreases the temp. inside the house to X11
Temp=X11
P8.1
E13
LightIntens =DxR
Increase/ decrease light intensity to match increase/ decrease of slider
LightIntens = Dimmer-Slider
P8.3
E14
Daylight=false Lights(X15)=on LightInten(X15) =DxR
Shut down the lights
Daylight=false Lights(X15)=off LightInten.(X14)=0
P2.2
E7
VacationControl=on, TV=off
Turn on TV for 60 min
VacationControl=on, TV=on
P8.4
E15
Daylight =True Lights=off
Turn lights on
Daylight =false, Lights=on
P2.3
VacationControl=on Lights(X5)=off
Turn on lights for 60 min. in X5
VacationControl=on, Lights(X5)=on
P9.2
Daylight = false/true, CurtainBlind (X18)= close/ open
Open/close curtains,blinds in X18
Daylight = true/false, CurtainBlind (X18)= open/ close
P4.2
X6On=true /false
Turn on/off the A/V device X6
X6On=true /false
P9.2
E16
Daylight = false/true, CurtainBlind (X18)= close/ open
Open/close curtains,blinds in X18
Daylight = true/false, CurtainBlind (X18)= open/ close
P6.2
Temp= DxR
Increase/ decrease temp of house to X12
Temp=X12
P11.1
E17
Tap/shower-valve=open
Shutdown water
Tap/shower Valve= close
P9.1
CurtainBlinds (x16) =close / open
Open/close the blinds and curtains in X16
CurtainBlinds (x16) =open/ close
P12.1
E18
Telephone =idle, RemoteAccess=idle
Activate remote access module
Telephone=busy RemoteAccess= Active
P10.1
Windows (X19)= close/open
Open/close windows in X19
Windows(X19) = open/close
P13.2
Telephone =idle, RemoteAccess=idle
Activate answer Machine
Telephone=busy AnswerMachine =Active
P14.1
Stove=DxR
Shutdown & prevent stove activation till X24
Stove=off
P15.1
E20
KitchenFan =off
Turn on kitchen fan
KitchenFan =on
P3.2
E9
MainDoor =closed, MainDoorLock =closed
Unlock and open the main door
MainDoor=open MainDoorLock =open
P15.2
E19
KitchenFan =on
Turn off kitchen fan
KitchenFan off=

Figure 8.4 - Continued: Trigger events chart of the smart homes dynamic behaviour policies

8.5.7 Step 6: Interaction Detection

8.5.7.1 Summary of the Detected Interactions

In this step, the developer detects interactions between simple policies using the interaction scenarios that are part of the basic core of IRIS (sixth step) or interaction scenarios that are inserted as plug-ins in the sixth step of IRIS. The developer tries to find interactions as explained in Section 4.3.7.

Table 8.6 presents the summary of all obtained results. The simple policy column contains the simple policy under investigation while the interacting simple policies column lists the simple policies that interact with the simple policy under investigation. Note that when an interaction is detected between two simple policies (e.g. P1.1 and P1.2), then this interaction is listed in the row of the first policy (P1.1) only and will not be repeated as part of the interactions of the second policy (P1.2). The total number of detected unique interactions is 83 interactions (as can been seen from the interactions in Table 8.6).

Table 8.6: Results summary of detected interactions among smart homes policies

Policy	Interacting policies	Policy	Interacting policies
P1.1	P1.2, P1.3, P1.4, P1.5, P3.2, P10.1, P12.1	P1.2	P3.2, P10.1, P12.1
P1.3	P3.2, P3.3, P12.1	P1.4	P3.2, P8.2, P12.1
P1.5	P3.2, P8.2, P12.1	P2.1	P2.2, P2.3, P10.1, P12.1
P2.2	P4.1, P4.2, P5.2, P9.1, P9.2, P10.1, P12.1	P2.3	P6.1, P6.2, P8.1, P8.2, P8.3, P8.4, P9.1, P9.2, P10.1, P12.1
P3.1	P3.3, P12.1	P3.2	P6.1, P6.2, P12.1
P3.3	P6.1, P6.2, P12.1	P4.1	P4.2, P5.1, P5.2, P12.1
P4.2	P5.2, P12.1	P5.1	P5.2, P12.1
P5.2	P12.1, P16.1	P6.1	P6.2, P10.1, P12.1
P6.2	P10.1, P12.1	P7.1	No Interactions
P7.2	No Interactions	P8.1	P8.2, P8.4, P12.1
P8.2	P12.1	P8.3	P12.1
P8.4	P12.1	P9.1	P9.2, P12.1
P9.2	P12.1	P10.1	P12.1
P11.1	P12.1	P12.1	P13.1, P13.2, P14.1, P14.2, P15.1, P16.1
P13.1	No Interactions	P13.2	No Interactions
P14.1	P16.1	P14.2	P16.1
P15.1	P16.1		

8.5.7.2 Interactions According to Main Interaction Category ①

This interaction main category contains interactions that occur between two system axioms. There are two interaction scenarios used to detect interactions under this main interactions category: SCR1 and SCR2. The developer is required to examine the system axioms table (see Table 8.2) developed in IRIS step 2. The developer has to pair-wise compare all system axioms with the aim of finding interactions according to either SCR1 or SCR2.

According to Table 8.2, there are 15 comparisons necessary in order to examine all system axioms in the smart homes case study. Table 8.7 provides an example of such an interaction that was detected. Full results of interactions between two system axioms are listed in Appendix E.

Table 8.7: Example of interaction between two system axioms using SCR1

Interaction ID	I52
Type of Interaction	Interaction between two system axioms
Interacting simple policies	P4.1 and P5.2
SCR used	SCR1
Explanation	There is a contradiction between the value of the *Rule* of P4.1 and the value of the rule attribute of P5.2. The rule of P4.1 can override the rule of P5.2 and vice versa. An interaction scenario can be "what happens when the user tries to use the remotes to go beyond the max audio level of the house?" If the system allows the user to use the remote control to exceed the maximum audio level then the P4.1 rule has overridden the P5.2 rule. But if the system will not allow the user to use the remote to go beyond the maximum audio level then the P5.2 rule has overridden P4.1 rule.

8.5.7.3 Interactions According to Main Interaction Categories ② and ⑦

The two main interaction categories ② and ⑦ have seven interaction scenarios: SCR3, SCR4, SCR5, SCR6, SCR7, SCR30, and SCR31. The developer has to compare pair-wise every dynamic behaviour simple policy with every system axiom with the objective of finding interactions according to any one of the seven interaction scenarios.

There are 174 comparisons necessary in order to detect all possible interactions between a system axiom and a dynamic behaviour simple policy in the case of the smart homes case study. Table 8.8 provides an example of an interaction detected between a system axiom simple policy and a dynamic behaviour simple policy. The full results of detected interactions are listed in Appendix E.

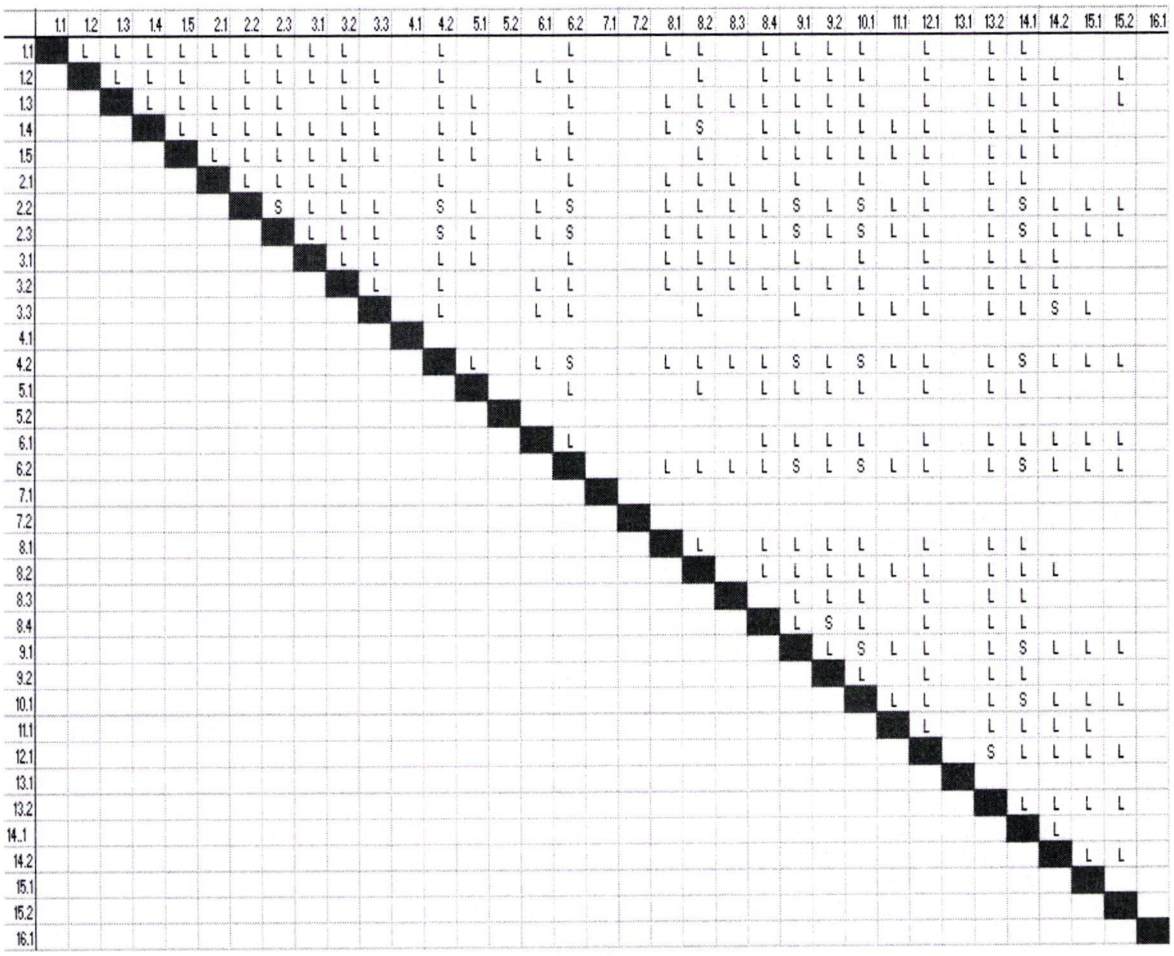
Table 8.8: Example of interaction between a system axiom and a dynamic behaviour simple policy using SCR30

Interaction ID	I24
Type of Interaction	Interaction between a dynamic behaviour simple policy and a system axiom simple policy
Interacting simple policies	P2.2 and P4.1
SCR used	SCR30
Explanation	There is a contradiction between the value of the *Action* attribute of the dynamic behaviour simple policy (P2.2) and the value of the *Rule* attribute for the system axiom (P4.1). The action of P2.2 overrides the rule of P4.1. A possible interaction scenario could be the following: A user gets home while the vacation control P2.2 is active and the action of it is being executed. The user tries to use the remote control to switch off the TV (P4.1). According to the definition of the vacation control P2.2, the control of the TV is now exclusively done by it and the remote control will not be able to switch off the TV. Hence, the action of P2.2 has overridden the rule of P4.1.

8.5.7.4 Interactions According to Main Interaction Category ③

The third interaction main category contains interactions that would occur between two dynamic behaviour simple policies. There are 5 basic core interaction scenarios used under this main interaction category: SCR8, SCR10, SCR11, SCR12, and SCR13. Moreover, there are 4 plug-ins interaction scenarios used under this interaction main category: SCR9, SCR14, SCR15, and SCR16. The developer compares every two dynamic behaviour simple policies that are triggered by the same trigger event or by linked trigger events.

There are 319 comparisons necessary in order to detect all possible interactions between two dynamic behaviour simple policies (25 comparisons resulting from examining dynamic behaviour simple policies triggered by the same trigger event plus 294 comparisons resulting from examining dynamic behaviour simple policies triggered by linked trigger events). Due to the large number of comparisons, they have been listed in Figure 8.5 where an L indicated linked events between the two simple-policies in the row and column of that cell. Similarly an S indicates that the two policies in the corresponding row and column simple policies are triggered by the same trigger event. The developer has now to analyze the dynamic behaviour simple policies pairs indicated in Figure 8.5 using the 9 interaction scenarios listed above.



	1.1	1.2	1.3	1.4	1.5	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	5.1	5.2	6.1	6.2	7.1	7.2	8.1	8.2	8.3	8.4	9.1	9.2	10.1	11.1	12.1	13.1	13.2	14.1	14.2	15.1	15.2	16.1
1.1	■	L	L	L	L	L	L	L	L	L			L				L			L	L		L	L	L	L		L		L	L				
1.2		■	L	L	L		L	L	L	L	L		L			L	L				L		L	L	L	L		L		L	L	L		L	
1.3			■	L	L	L	L	L		L	L		L	L			L			L	L	L	L	L	L	L		L		L	L	L		L	
1.4				■	L	L	L	L	L	L	L		L	L			L			L	S		L	L	L	L	L	L		L	L	L			
1.5					■	L	L	L	L	L	L		L	L		L	L				L		L	L	L	L	L	L		L	L	L			
2.1						■	L	L	L	L			L				L			L	L	L		L		L		L		L	L				
2.2							■	S	L	L	L		S	L		L	S			L	L	L	L	S	L	S	L	L		L	S	L	L	L	
2.3								■	L	L	L		S	L		L	S			L	L	L	L	S	L	S	L	L		L	S	L	L	L	
3.1									■	L	L		L	L			L			L	L	L		L		L		L		L	L	L			
3.2										■	L		L			L	L			L	L	L	L	L	L	L		L		L	L	L			
3.3											■		L			L	L				L			L		L	L	L		L	L	S	L		
4.1												■																							
4.2													■	L		L	S			L	L	L	L	S	L	S	L	L		L	S	L	L	L	
5.1														■			L				L		L	L	L	L		L		L	L				
5.2															■																				
6.1																■	L						L	L	L	L		L		L	L	L	L	L	
6.2																	■			L	L	L	L	S	L	S	L	L		L	S	L	L	L	
7.1																		■																	
7.2																			■																
8.1																				■	L		L	L	L	L		L		L	L				
8.2																					■		L	L	L	L	L	L		L	L	L			
8.3																						■		L	L	L		L		L	L				
8.4																							■	L	S	L		L		L	L				
9.1																								■	L	S	L	L		L	S	L	L	L	
9.2																									■	L		L		L	L				
10.1																										■	L	L		L	S	L	L	L	
11.1																											■	L		L	L	L	L		
12.1																												■		S	L	L	L	L	
13.1																													■						
13.2																														■	L	L	L	L	
14.1																															■	L			
14.2																																■	L	L	
15.1																																	■		
15.2																																		■	
16.1																																			■

Figure 8.5: List of the comparisons needed to detect interactions between dynamic behaviour simple policies in the smart homes case study

Table 8.9 provides an example of a detected interaction between two dynamic behaviour policies triggered by the same trigger event and Table 8.10 provides an example of a detected interaction between two dynamic behaviour simple policies triggered by linked trigger events. All other interactions between two dynamic behaviour simple policies are listed in Appendix E.

Table 8.9: Example of interaction between two dynamic behaviour simple policies triggered by the same trigger event using SCR11

Interaction ID	I63
Type of Interaction	Interaction between Two Dynamic Behaviour Simple Policies
Interacting simple policies	P6.2 and P10.1
SCR used	SCR11
Explanation	Both simple policies are triggered by E7 AND they have the same pre-states AND there is a negative impact between the two actions of the two simple policies. The action of P10.1 has a negative impact on the action of P6.2. An example of an interaction scenario: "The system opens the windows and at the same time tries to raise the temperature of the house". It is obvious that if the temperature outside the house is too cold (or too hot) then action of P10.1 has negative impact on action of P6.2.

Table 8.10: Example of interaction between two dynamic behaviour simple policies triggered by linked trigger event using SCR12

Interaction ID	I65
Type of Interaction	Interaction between Two Dynamic Behaviour Simple Policies
Interacting simple policies	P8.1 and P8.2
SCR used	SCR12
Explanation	These two simple policies are triggered by the linked events E4 and E13 where E4 ~> E13. The action of P8.1 overrides and cancels the action of P8.2 before its completion. An example of an interaction scenario is the situation when someone wakes up at night and tries to increase the light through the light dimmer. According to P8.2 a person that wakes up at night and walks into a specified part of the house causes the lights to increase in that part to a maximum over the period of two minutes. But the user can turn the light dimmer after 30 seconds to increase/decrease the light intensity. In such a situation, P8.2 was triggered first by E13, i.e., the system starts increasing the light over a period of two minutes. But then that person changes the light dimmer to increase/decrease the lights and thus triggering P8.1 which in turn overrides the action of P8.1 before its completion.

8.6 Discussion of the Results

8.6.1 Reduction in Number of Comparisons

IRIS required the following 508 comparisons to be done:

- 15 comparisons necessary to detect interactions according to main interaction category 1 (number of all possible pair-wise comparisons according to Table 8.2)
- 174 comparisons necessary to detect all possible interactions according to main interaction categories 2 and 7 (number of all possible pair-wise comparisons according to Table 8.2 and Table 8.3)
- 319 comparisons necessary to detect interactions according to main interaction category 3 as discussed in Section 8.5.7.4.

A human expert, however, would need 630 comparisons to pair-wise compare all simple policies of the smart homes case study. Thus we have achieved a 19.3% reduction in the number of pair-wise comparisons.

8.6.2 Comparing IRIS Results with Other Results Reported in the Literature

The smart homes case study had the following number of features and simple policies as presented in Table 8.11:

Table 8.11: statistics on the smart homes case study

Number of Features	16
Number of simple policies	35
Number of detected interactions using IRIS	83

This section evaluates the results from applying IRIS in the smart homes case study. Accuracy shows how precise was the detected interactions and if any interactions were missed. Unfortunately, there are no fully documented results in the literature with which we could have compared our results. However, Kolberg *et al.* in [21] lists an overview of some interactions that arise between services supporting networked appliances in a smart home environment. This overview included some interaction examples. All interaction examples mentioned in [21] were detected using IRIS.

8.7 Summary

This chapter proposed the use of the IRIS semi-formal approach to detect interactions in the smart homes domain. A comprehensive view of the distinction between policies and features was presented. A case study was carried out for detecting interactions among policies in smart homes using the proposed semi-formal approach and was presented in this chapter. The proposed approach was successfully customized and applied in the smart homes domain. It was able to detect 83 interactions among 35 user policies using only 508 pair-wise comparisons as apposed to 630 a human expert would have to do and thus achieving a reduction of 19.3% in the number of comparisons. These results support the chapter's main claim of being able to use our semi-formal approach, IRIS, to successfully detect interactions between policies. Further, these results serve as the first fully documented results of interactions between policies in the smart homes domain.

CHAPTER NINE: IRIS TOOL SUPPORT

9.1 Introduction

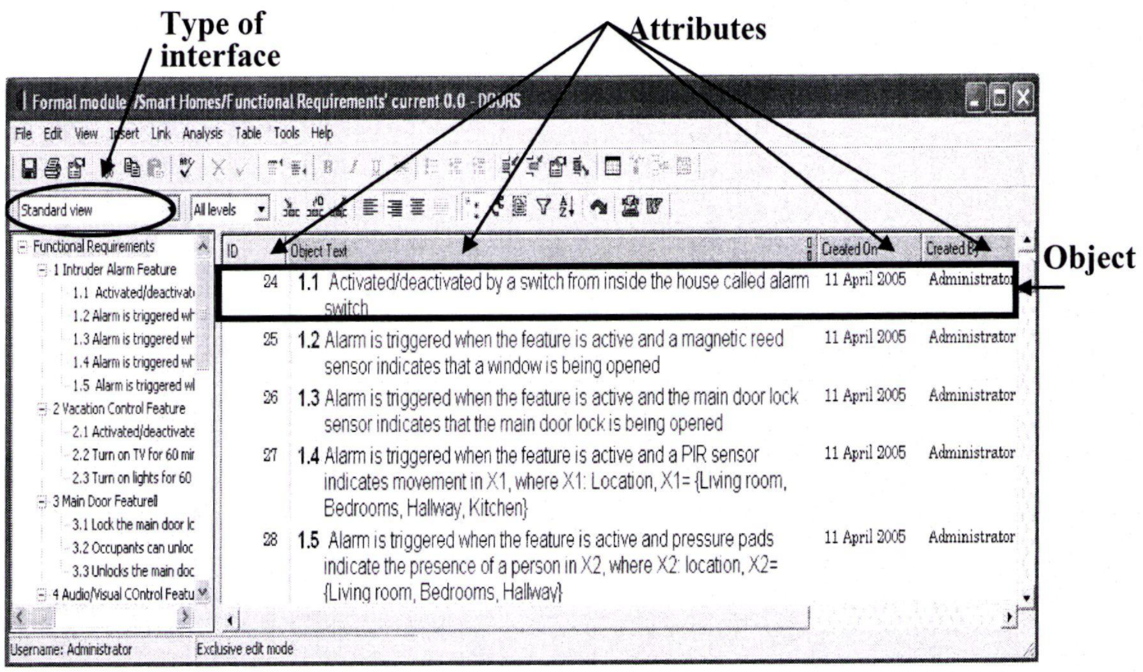
This chapter introduces IRIS-TS which stands for Identifying Requirements Interactions using Semi-formal methods -Tool Support. IRIS-TS is a tool support for applying IRIS to detect interactions between a set of requirements. For this reason, IRIS-TS was designed and implemented as an add-on that can be added to DOORS [23] which is one of the most famous and commonly used requirements management tools in both academia and industry.

This chapter presents the general architecture of IRIS-TS. Section 9.2 describes a general overview of the architecture and design of IRIS-TS. Section 9.3 then presents an overview of a prototype that was created for IRIS-TS in DOORS. This section also includes screen shots taken from applying the IRIS-TS prototype on the smart homes case study. Finally, Section 9.4 presents the summary of this chapter.

9.2 Architecture of IRIS-TS

IRIS-TS is a tool support that is implemented as independent code files that can be inserted as an add-on to DOORS to facilitate the detection of requirements interactions using IRIS. DOORS is one of the most commonly used requirements management tools for documenting and managing requirements for software systems. However, DOORS does not have any sort of interaction detection support built in it. IRIS-TS is implemented to be installed as an add-on to extend DOORS to support requirements interaction detection using IRIS. In this section, we focus first on describing the general architecture of IRIS-TS.

DOORS consists of modules that contain data and interfaces to show the data contents of these modules. A module is the way that DOORS uses to store data. A module is like a sheet on which data is written and stored. For example, in a specific software system that uses DOORS, there will be a module that contains all the system requirements and a module that contains all the tests for validating the final product. Each module consists of Objects and Attributes which corresponds to rows and columns, respectively. Objects and attributes are used to represent the information stored within a module. For example, the requirements module will contain an object (row) O1 that represents a requirement R1. The object O1 has attributes that describe requirement R1 such as ID, Object text, Created by, and Modified on. These attributes are part of the default set of attributes that comes with DOORS. Figure 9.1 shows a screen shot of the module "Functional Requirements" that has the requirements of smart homes and how the requirements are stored in the module as objects (rows) and attributes (columns).



Formal module '/Smart Homes/Functional Requirements' current 0.0 - DOORS

- Functional Requirements
  - 1 Intruder Alarm Feature
    - 1.1 Activated/deactivat
    - 1.2 Alarm is triggered wh
    - 1.3 Alarm is triggered wh
    - 1.4 Alarm is triggered wh
    - 1.5 Alarm is triggered wl
  - 2 Vacation Control Feature
    - 2.1 Activated/deactivate
    - 2.2 Turn on TV for 60 mir
    - 2.3 Turn on lights for 60
  - 3 Main Door Featurell
    - 3.1 Lock the main door lc
    - 3.2 Occupants can unloc
    - 3.3 Unlocks the main doc
  - 4 Audio/Visual COntrol Featu

ID	Object Text	Created On	Created By
24	1.1 Activated/deactivated by a switch from inside the house called alarm switch	11 April 2005	Administrator
25	1.2 Alarm is triggered when the feature is active and a magnetic reed sensor indicates that a window is being opened	11 April 2005	Administrator
26	1.3 Alarm is triggered when the feature is active and the main door lock sensor indicates that the main door lock is being opened	11 April 2005	Administrator
27	1.4 Alarm is triggered when the feature is active and a PIR sensor indicates movement in X1, where X1: Location, X1= {Living room, Bedrooms, Hallway, Kitchen}	11 April 2005	Administrator
28	1.5 Alarm is triggered when the feature is active and pressure pads indicate the presence of a person in X2, where X2: location, X2= {Living room, Bedrooms, Hallway}	11 April 2005	Administrator

Username: Administrator Exclusive edit mode

Figure 9.1: An example of modules, objects, and attributes

The interface, or sometimes called the view, is the graphical representation that DOORS uses to display the contents of the modules to the user. For example, in Figure 9.1, DOORS uses the interface type "standard view" to display the contents of the module Functional Requirement. Of course the interface itself is the whole screen.

The concept of attributes has been used in IRIS-TS to correspond to the requirements attributes that are used as part of IRIS. Only this time, new customized attributes are created in DOORS through the IRIS-TS code to represent the requirements attributes being used in IRIS. For example, IRIS-TS when executed will create new attributes that are applicable for all modules of DOORS such as "PreState", "Action", and "NextState". The concept of modules has been used by IRIS-TS to represent the tables and graphs that are generated through the different steps of IRIS. For example, IRIS-TS will create a module called "Trigger Events Extraction Table" to correspond to the table created in the trigger events extraction step in IRIS.

This brief introduction was important to understand the architecture of IRIS-TS which is shown in Figure 9.2. As can be seen, the IRIS-TS is implemented as an add-on that can be integrated into DOORS and communicate with its modules and interfaces. On the other hand, the analysts does not have to deal with IRIS-TS code or the creation of the new modules but rather he deals with the interfaces that are either used to display the contents of the modules or created by IRIS-TS to display/request data for/from analyst.

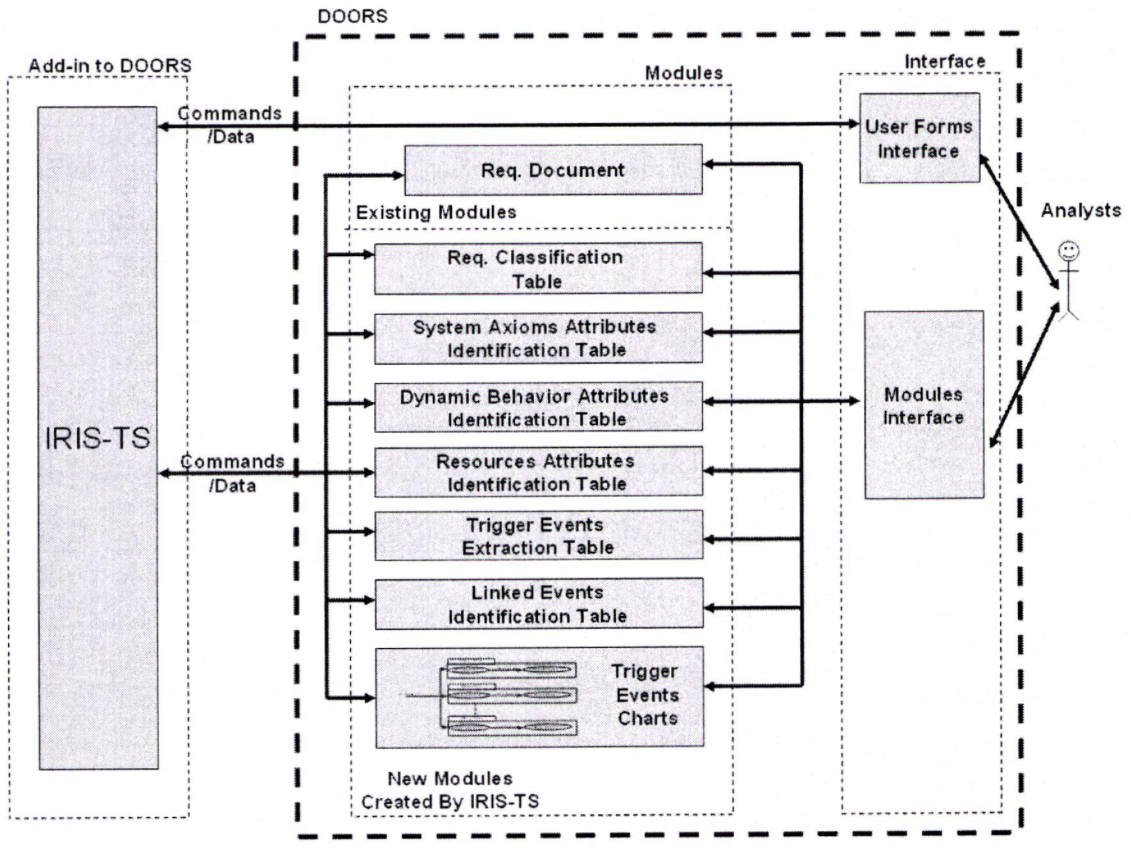

DOORS
Add-in to DOORS
Modules
Interface
IRIS-TS
Commands /Data
Commands /Data
User Forms Interface
Req. Document
Existing Modules
Req. Classification Table
System Axioms Attributes Identification Table
Dynamic Behavior Attributes Identification Table
Resources Attributes Identification Table
Trigger Events Extraction Table
Linked Events Identification Table
Trigger Events Charts
New Modules Created By IRIS-TS
Modules Interface
Analysts

Figure 9.2: Architecture of IRIS-TS

The architecture of IRIS-TS can be explained in a high abstraction level as follows: When the IRIS-TS is executed it will communicate with the module that has the requirements of the system stored in it (which is shown in Figure 9.2 as the module Req. Document). The communication carried out with the Req. Document module will be in the form of data regarding the requirements stored in this module or commands to create new attributes for requirements and store data in these attributes. IRIS-TS will also communicate with the analyst to interactively execute the different steps of IRIS in an ordered manner. During the execution of the different steps of IRIS, the tool will create customized interfaces (windows) that either display or request data to/from the analyst. The tool will also create new modules and assign customized attributes to these modules to store the data obtained after the execution of each step of IRIS.

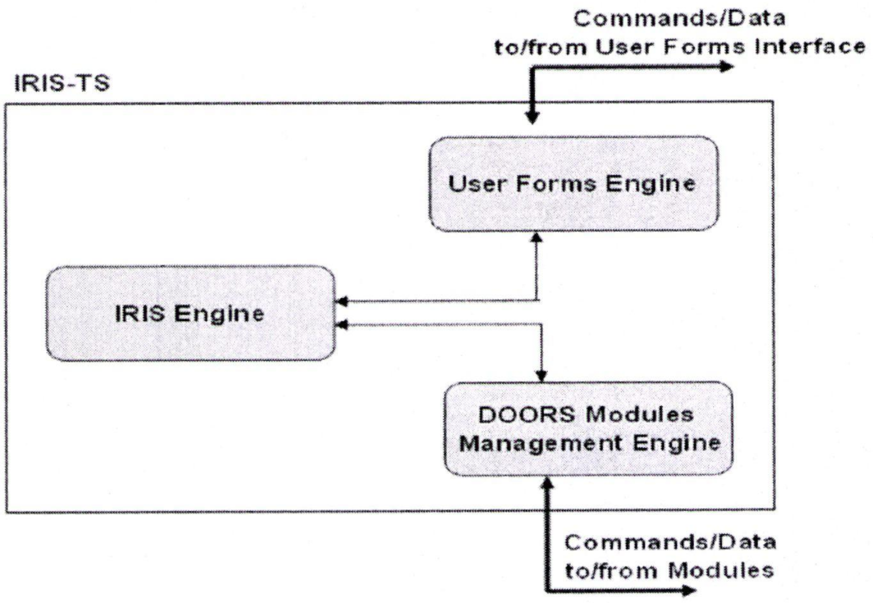


Figure 9.3: Internal structure of IRIS-TS

The way IRIS-TS manages these tasks is through the three engines in its interior as shown in Figure 9.3. The IRIS Engine is the main engine in the tool and is responsible for determining the next IRIS step to be carried out and what exactly needs to be done. If the IRIS step being executed requires data or creation of attributes to store data in the main Req. Document, then the IRIS engine will communicate with the Req. Document module ,through the "DOORS Modules Management Engine", to request data or create attributes to store specific data. If the IRIS step being executed requires communicating with the analysts to request or display data, then the IRIS engine will request that the User Forms Engine creates an interface with the necessary data and/or fields that needs to be entered by the analyst. The User Forms Engine will create the requested interface and send it to the interfaces part of IRIS to be displayed to the analyst. Once the analyst provides the adequate response, then that response and the data collected, when required, will be returned to the IRIS engine to determine what needs to be done. The IRIS engine after the execution of a complete step of IRIS will request from the DOORS Modules Management Engine to create new modules to store the data/tables/graphs that was created so that they can be used later. The DOORS Module Management engine, once it receives a request for module creation or data manipulation in a specific module, will issue the appropriate DOORS commands to carry out the request it receives.

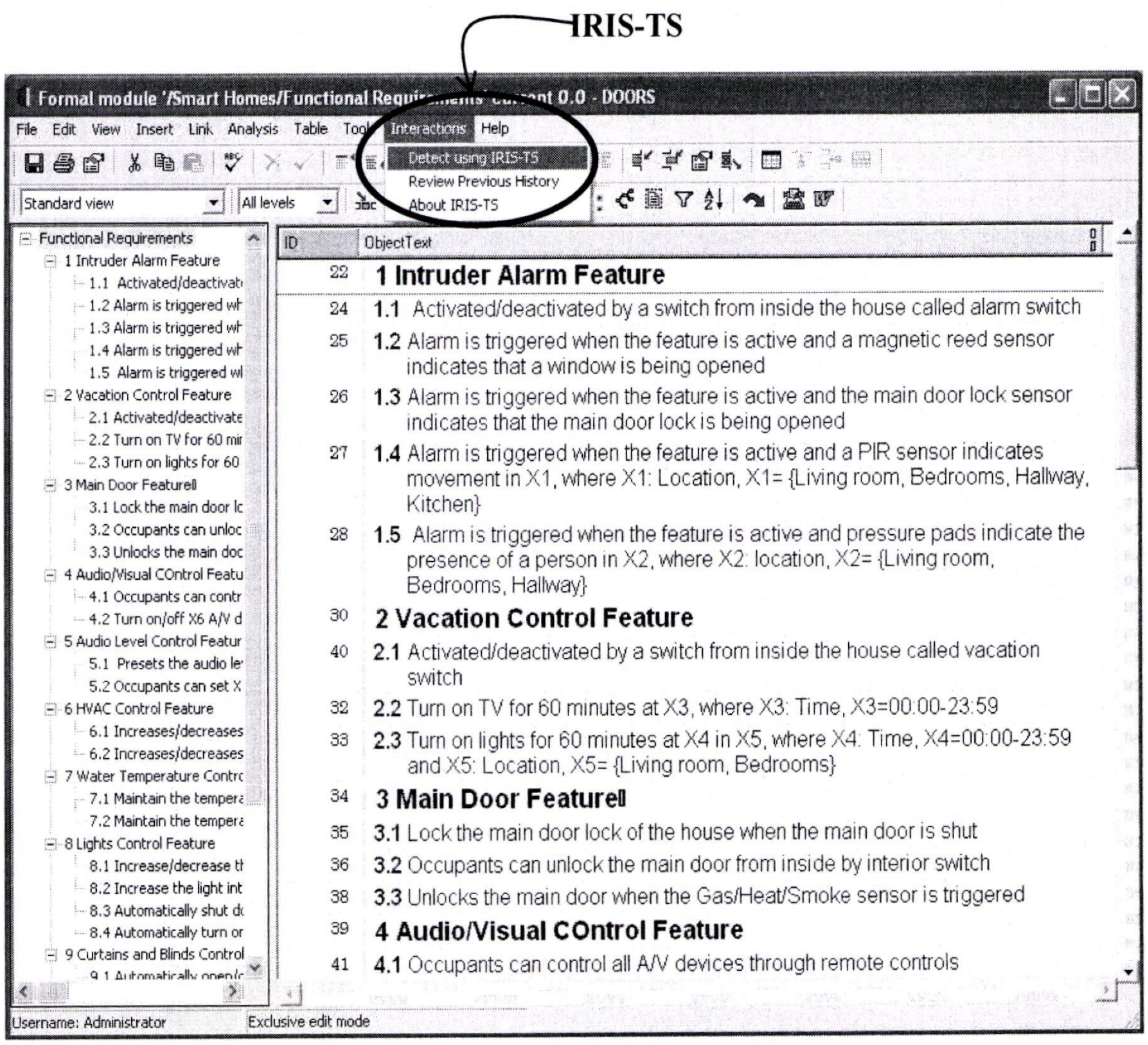
9.3 A prototype of IRIS-TS

9.3.1 Implementation

As described in section 9.2, IRIS-TS was created as an add-on to DOORS. The IRIS-TS tool was programmed using DOORS programming language DXL (DOORS eXtension Language). The programming language DXL is a scripting language specially developed for DOORS. DXL can be used provide many features, such as file format importers and exporters, impact and traceability analysis and inter-module linking tools. DXL can also be used to develop larger add-on packages such as IRIS-TS presented in this chapter. This capability to extend or customize DOORS is available to users who choose to develop their own DXL scripts. The DXL language is based on an underlying programming language whose fundamental data types, functions and syntax are largely based on C and C++. To support the needs of script writing, there are some differences. In particular, concepts like main program are avoided, and mandatory semicolons and parentheses have been discarded.

The way DXL is used is to either enter individual scripts in a specific window in DOORS and run these scripts to see how they work, or the other alternative would be to develop an add-on package that can be added to DOORS and with some specific scripts the DXL script can appear as a menu on the top bar of DOORS. In this thesis, IRIS-TS was developed as a complete add-on package that needs only to be installed in the add-on subdirectory located inside the DOORS main installation directory. Figure 9.4 shows how IRIS when installed as an add-on package would appear as a drop down menu in DOORS.

To give an example of the implementation of IRIS-TS using DXL, Appendix F presents part of the DXL code for executing the first step of IRIS to give a feeling of how the DXL code, that was written for IRIS-TS, looks like. It is worth mentioning that the complete DXL code of the tool is more than 70 pages using the format of Appendix F.



ID	ObjectText
22	**1 Intruder Alarm Feature**
24	**1.1** Activated/deactivated by a switch from inside the house called alarm switch
25	**1.2** Alarm is triggered when the feature is active and a magnetic reed sensor indicates that a window is being opened
26	**1.3** Alarm is triggered when the feature is active and the main door lock sensor indicates that the main door lock is being opened
27	**1.4** Alarm is triggered when the feature is active and a PIR sensor indicates movement in X1, where X1: Location, X1= {Living room, Bedrooms, Hallway, Kitchen}
28	**1.5** Alarm is triggered when the feature is active and pressure pads indicate the presence of a person in X2, where X2: location, X2= {Living room, Bedrooms, Hallway}
30	**2 Vacation Control Feature**
40	**2.1** Activated/deactivated by a switch from inside the house called vacation switch
32	**2.2** Turn on TV for 60 minutes at X3, where X3: Time, X3=00:00-23:59
33	**2.3** Turn on lights for 60 minutes at X4 in X5, where X4: Time, X4=00:00-23:59 and X5: Location, X5= {Living room, Bedrooms}
34	**3 Main Door Featurell**
35	**3.1** Lock the main door lock of the house when the main door is shut
36	**3.2** Occupants can unlock the main door from inside by interior switch
38	**3.3** Unlocks the main door when the Gas/Heat/Smoke sensor is triggered
39	**4 Audio/Visual COntrol Feature**
41	**4.1** Occupants can control all A/V devices through remote controls

Figure 9.4: IRIS-TS implementation in DOORS

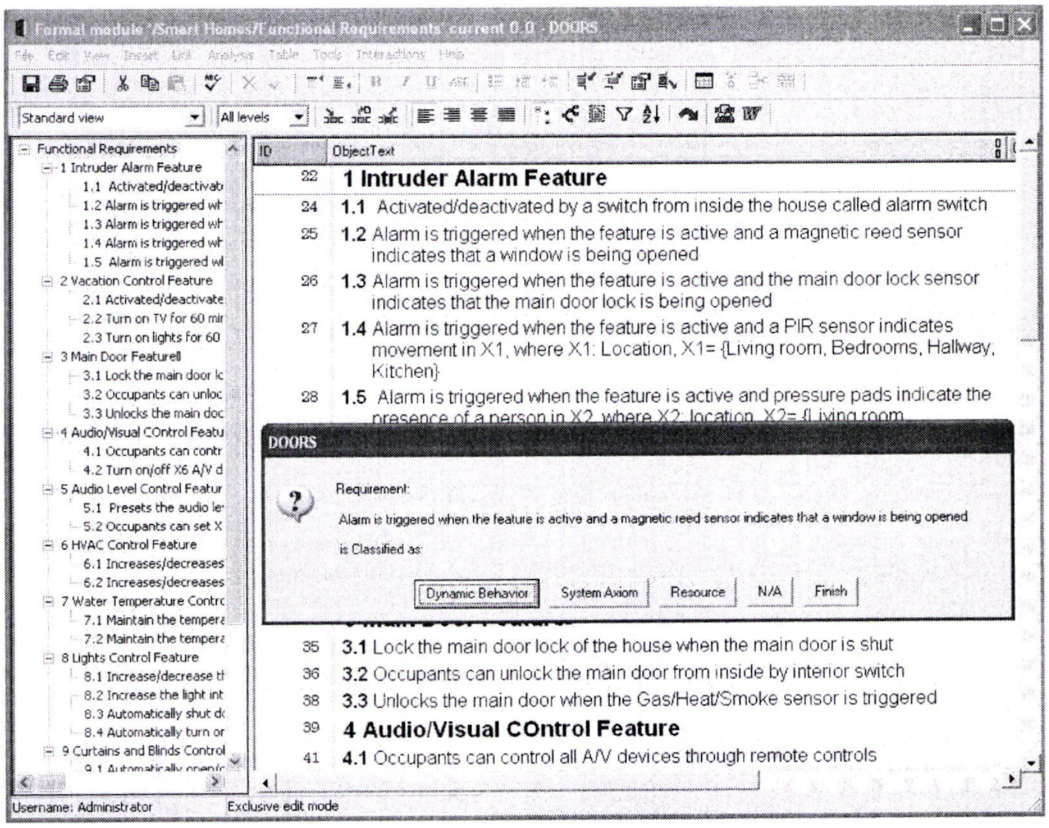
9.3.2 Applying the IRIS-TS Prototype on the Smart Homes Case Study

To demonstrate and describe the IRIS-TS when it is executed in the DOORS environment on an actual requirement document, this section presents screen shots taken from applying IRIS-TS on the smart homes case study that was presented previously in Chapter 8. For each step of IRIS, two screen shots are presented. The first screenshot shows how IRIS-TS performs the IRIS step being executed. The second screenshot shows the result that IRIS-TS has generated from performing the IRIS step being considered. As can be seen from Figure 9.4, the developer will have to open the Interactions drop down menu and choose "Detect using IRIS-TS". This will execute IRIS-TS code to detect interactions between the requirements of the smart homes. The requirements of the smart homes are stored as objects. Each requirement will have an ID attribute which uniquely identifies it and an object text attribute that contains the textual description of the requirement. In the following we present the screenshots of applying IRIS-TS to detect interactions.

9.3.2.1 Requirements Classification using IRIS-TS

The first step of IRIS is requirements classification into system axioms, dynamic behaviour requirements, or resources. This step is carried out as shown in Figure 9.5. IRIS-TS will display a message for each requirement and ask the analyst to classify it as a system axiom or a dynamic behaviour requirement or a resource.

Once finished displaying all requirements to the analyst to be classified, IRIS-TS will create a new module to correspond to the requirements classification table created in IRIS step 1 (see Section 4.3.2). The created module contains all the requirements along with their classification stored in an attribute called classification. The results of this table is shown in Figure 9.6



ID	ObjectText
22	**1 Intruder Alarm Feature**
24	**1.1** Activated/deactivated by a switch from inside the house called alarm switch
25	**1.2** Alarm is triggered when the feature is active and a magnetic reed sensor indicates that a window is being opened
26	**1.3** Alarm is triggered when the feature is active and the main door lock sensor indicates that the main door lock is being opened
27	**1.4** Alarm is triggered when the feature is active and a PIR sensor indicates movement in X1, where X1: Location, X1= {Living room, Bedrooms, Hallway, Kitchen}
28	**1.5** Alarm is triggered when the feature is active and pressure pads indicate the presence of a person in X2, where X2: location, X2= {Living room
35	**3.1** Lock the main door lock of the house when the main door is shut
36	**3.2** Occupants can unlock the main door from inside by interior switch
38	**3.3** Unlocks the main door when the Gas/Heat/Smoke sensor is triggered
39	**4 Audio/Visual COntrol Feature**
41	**4.1** Occupants can control all A/V devices through remote controls

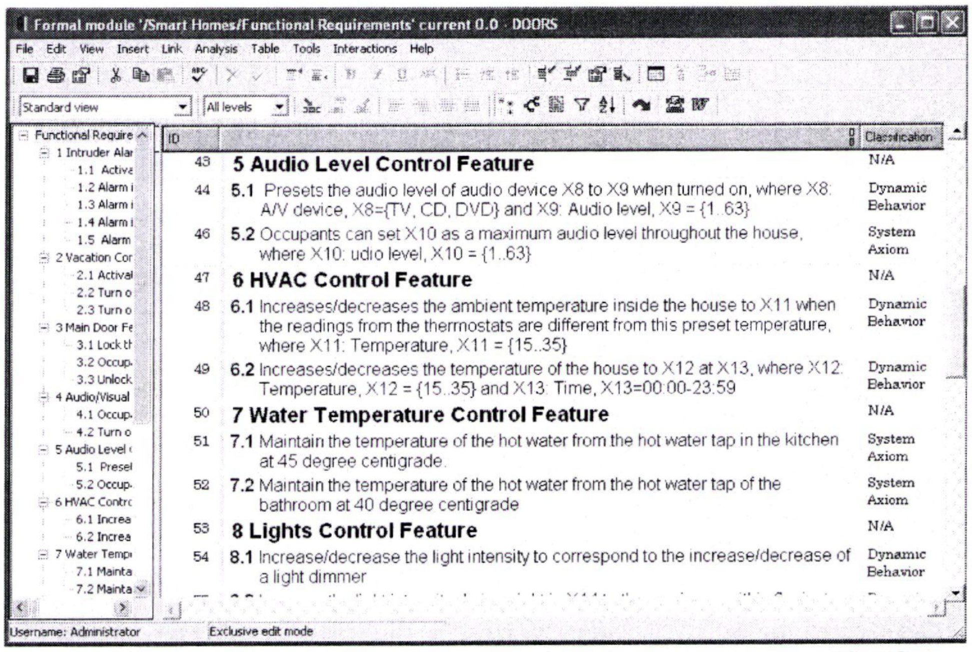
DOORS

Requirement:

Alarm is triggered when the feature is active and a magnetic reed sensor indicates that a window is being opened

is Classified as:

Figure 9.5: Performing requirements classification using IRIS-TS



ID		Classification
43	**5 Audio Level Control Feature**	N/A
44	**5.1** Presets the audio level of audio device X8 to X9 when turned on, where X8: A/V device, X8={TV, CD, DVD} and X9: Audio level, X9 = {1..63}	Dynamic Behavior
46	**5.2** Occupants can set X10 as a maximum audio level throughout the house, where X10: udio level, X10 = {1..63}	System Axiom
47	**6 HVAC Control Feature**	N/A
48	**6.1** Increases/decreases the ambient temperature inside the house to X11 when the readings from the thermostats are different from this preset temperature, where X11: Temperature, X11 = {15..35}	Dynamic Behavior
49	**6.2** Increases/decreases the temperature of the house to X12 at X13, where X12: Temperature, X12 = {15..35} and X13: Time, X13=00:00-23:59	Dynamic Behavior
50	**7 Water Temperature Control Feature**	N/A
51	**7.1** Maintain the temperature of the hot water from the hot water tap in the kitchen at 45 degree centigrade.	System Axiom
52	**7.2** Maintain the temperature of the hot water from the hot water tap of the bathroom at 40 degree centigrade	System Axiom
53	**8 Lights Control Feature**	N/A
54	**8.1** Increase/decrease the light intensity to correspond to the increase/decrease of a light dimmer	Dynamic Behavior

Figure 9.6: Results of requirements classification using IRIS-TS

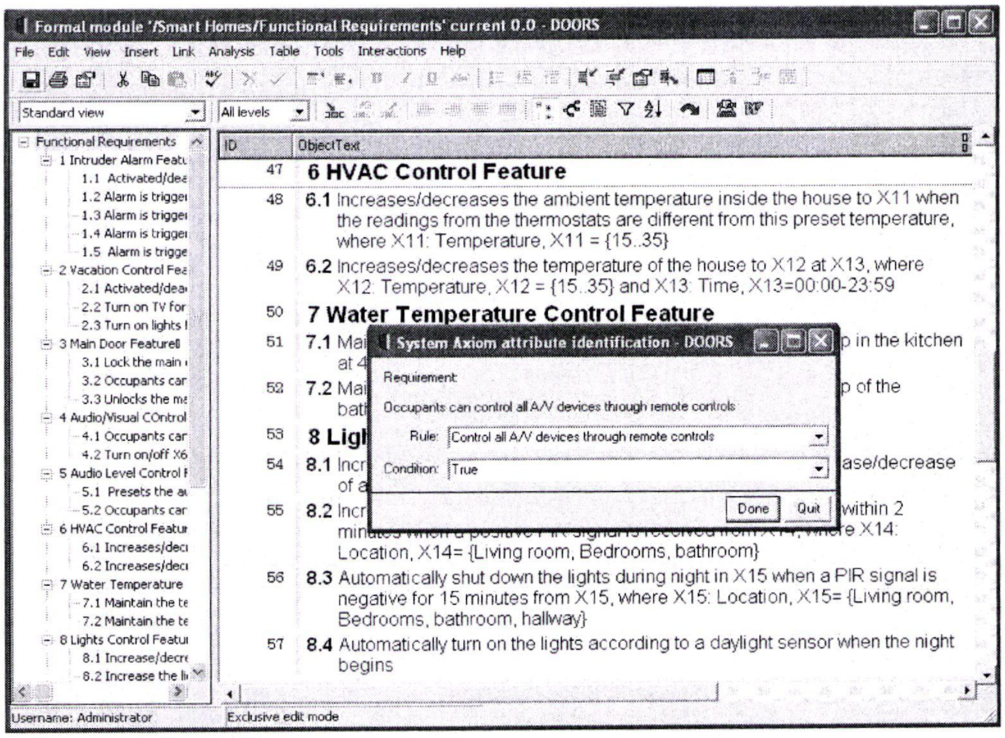
9.3.2.2 Requirements Attributes Identification using IRIS-TS

The second step of IRIS is attributes identification for system axioms, dynamic behaviour requirements and resources. In the smart homes case study, IRIS-TS will start displaying messages to the analyst asking him to identify the attributes of the system axioms first (Rule and Condition) and then to identify the attributes of the dynamic behaviour requirements (Prestate, Trigger Event, Action, and Next State). Note that the attributes ID and Description are obtained automatically for the original requirement module. Also, the attributes Parameters and Parameters range were not implemented in IRIS-TS.

For the sake of aiding the analyst to be consistent in using the same terminologies for defining the attributes, a drop-down buffer is available with each attribute to list all previously entered attributes during the execution of IRIS-TS.

Once IRIS-TS finishes all requirements attributes identification, it will create two new modules with attributes corresponding the requirements attributes to save all the data collected from the analyst. These two modules correspond to the system axiom attributes identification table and dynamic behaviour attributes identification table created in step 2 of IRIS (see Section 4.3.3).

Figures 9.7 and 9.8 shows screenshots for messages to the analyst to enter values for the system axiom and dynamic behaviour requirements attributes respectively. Figures 9.9 and 9.10 shows screenshots for the created modules for the system axiom attributes identification and dynamic behaviour attributes identification, respectively.



47 **6 HVAC Control Feature**

48 **6.1** Increases/decreases the ambient temperature inside the house to X11 when the readings from the thermostats are different from this preset temperature, where X11: Temperature, X11 = {15..35}

49 **6.2** Increases/decreases the temperature of the house to X12 at X13, where X12: Temperature, X12 = {15..35} and X13: Time, X13=00:00-23:59

50 **7 Water Temperature Control Feature**

51 **7.1** Mai ... p in the kitchen at 4

52 **7.2** Mai ... p of the bat

53 **8 Ligl**

54 **8.1** Incr ... ase/decrease of a

55 **8.2** Incr ... within 2 min ... re X14: Location, X14= {Living room, Bedrooms, bathroom}

56 **8.3** Automatically shut down the lights during night in X15 when a PIR signal is negative for 15 minutes from X15, where X15: Location, X15= {Living room, Bedrooms, bathroom, hallway}

57 **8.4** Automatically turn on the lights according to a daylight sensor when the night begins

System Axiom attribute identification - DOORS

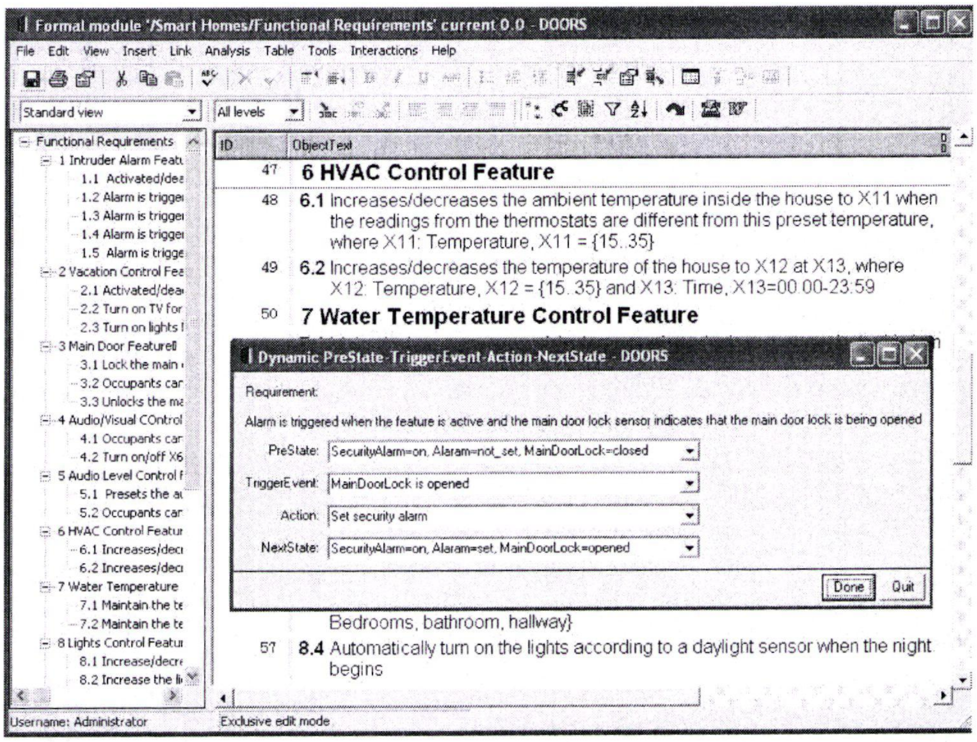
Requirement:

Occupants can control all A/V devices through remote controls

Rule: Control all A/V devices through remote controls

Condition: True

Figure 9.7: Performing system axioms attributes identification using IRIS-TS



Formal module '/Smart Homes/Functional Requirements' current 0.0 - DOORS

ID	ObjectText
47	**6 HVAC Control Feature**
48	**6.1** Increases/decreases the ambient temperature inside the house to X11 when the readings from the thermostats are different from this preset temperature, where X11: Temperature, X11 = {15..35}
49	**6.2** Increases/decreases the temperature of the house to X12 at X13, where X12: Temperature, X12 = {15..35} and X13: Time, X13=00.00-23:59
50	**7 Water Temperature Control Feature**
	Bedrooms, bathroom, hallway}
57	**8.4** Automatically turn on the lights according to a daylight sensor when the night begins

Dynamic PreState-TriggerEvent-Action-NextState - DOORS

Requirement:

Alarm is triggered when the feature is active and the main door lock sensor indicates that the main door lock is being opened

PreState: SecurityAlarm=on, Alaram=not_set, MainDoorLock=closed

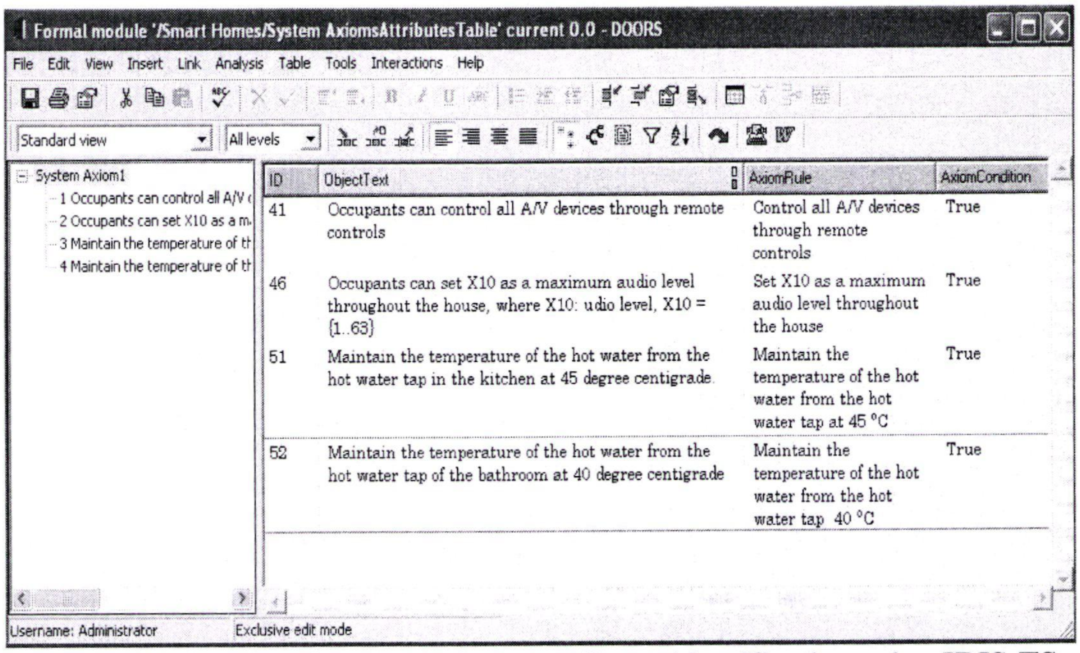
TriggerEvent: MainDoorLock is opened

Action: Set security alarm

NextState: SecurityAlarm=on, Alaram=set, MainDoorLock=opened

Username: Administrator Exclusive edit mode

Figure 9.8: Performing dynamic behaviour attributes identification using IRIS-TS

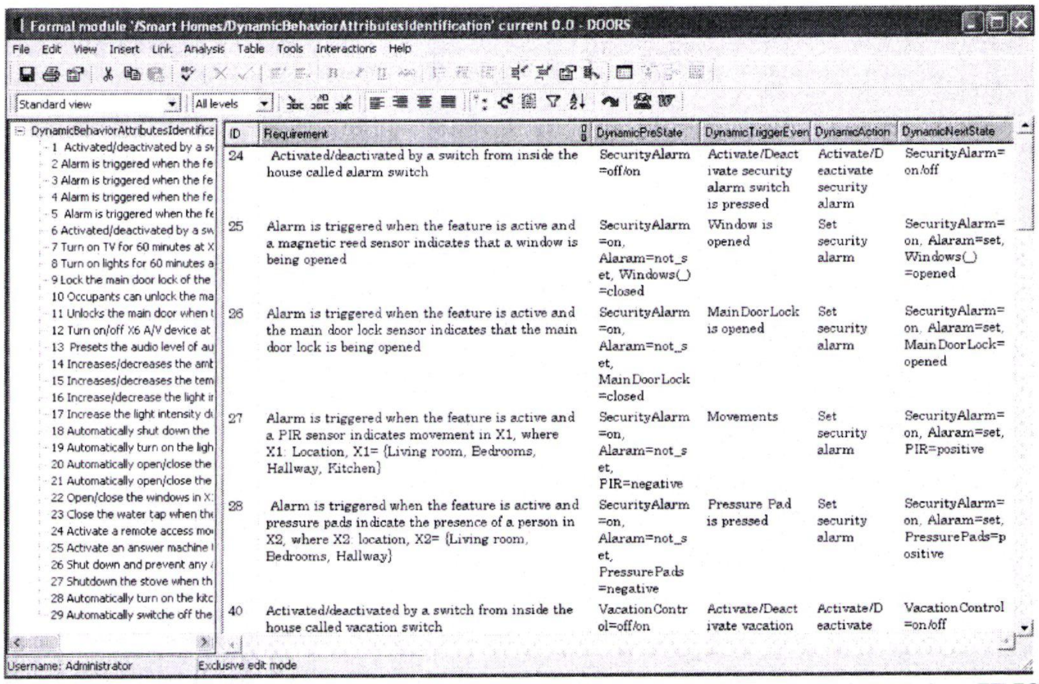


Formal module '/Smart Homes/System AxiomsAttributesTable' current 0.0 - DOORS

ID	ObjectText	AxiomRule	AxiomCondition
41	Occupants can control all A/V devices through remote controls	Control all A/V devices through remote controls	True
46	Occupants can set X10 as a maximum audio level throughout the house, where X10: udio level, X10 = {1..63}	Set X10 as a maximum audio level throughout the house	True
51	Maintain the temperature of the hot water from the hot water tap in the kitchen at 45 degree centigrade.	Maintain the temperature of the hot water from the hot water tap at 45 °C	True
52	Maintain the temperature of the hot water from the hot water tap of the bathroom at 40 degree centigrade	Maintain the temperature of the hot water from the hot water tap 40 °C	True

Username: Administrator Exclusive edit mode

Figure 9.9: Results of system axioms attributes identification using IRIS-TS


Formal module '/Smart Homes/DynamicBehaviorAttributesIdentification' current 0.0 - DOORS

File Edit View Insert Link Analysis Table Tools Interactions Help

Standard view | All levels

DynamicBehaviorAttributesIdentifica
- 1 Activated/deactivated by a sw
- 2 Alarm is triggered when the fe
- 3 Alarm is triggered when the fe
- 4 Alarm is triggered when the fe
- 5 Alarm is triggered when the fe
- 6 Activated/deactivated by a sw
- 7 Turn on TV for 60 minutes at X
- 8 Turn on lights for 60 minutes a
- 9 Lock the main door lock of the
- 10 Occupants can unlock the ma
- 11 Unlocks the main door when t
- 12 Turn on/off X6 A/V device at
- 13 Presets the audio level of au
- 14 Increases/decreases the amb
- 15 Increases/decreases the tem
- 16 Increase/decrease the light ir
- 17 Increase the light intensity du
- 18 Automatically shut down the
- 19 Automatically turn on the ligh
- 20 Automatically open/close the
- 21 Automatically open/close the
- 22 Open/close the windows in X
- 23 Close the water tap when the
- 24 Activate a remote access mo
- 25 Activate an answer machine
- 26 Shut down and prevent any
- 27 Shutdown the stove when th
- 28 Automatically turn on the kitc
- 29 Automatically switche off the

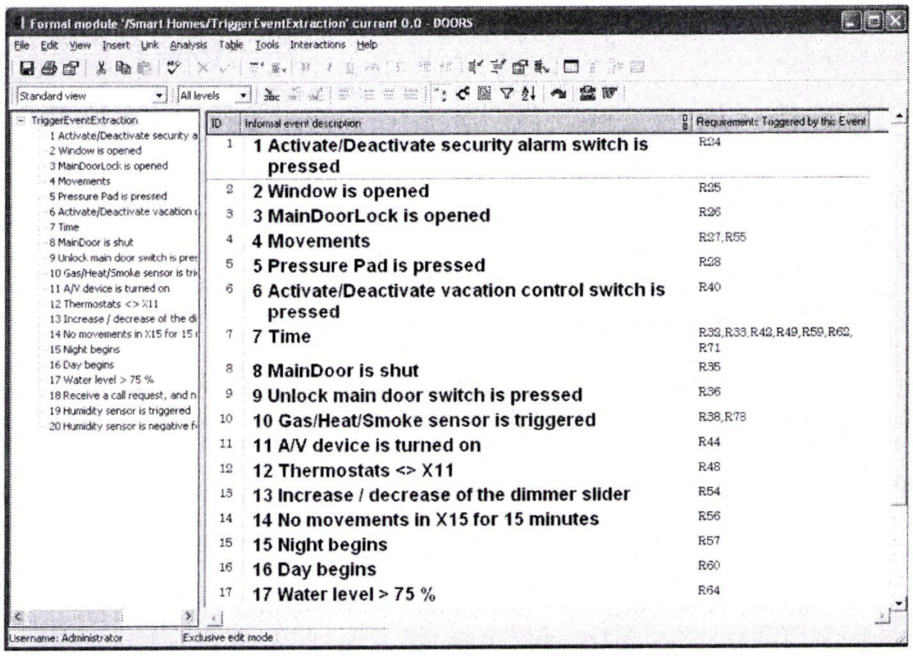
ID	Requirement	DynamicPreState	DynamicTriggerEven	DynamicAction	DynamicNextState
24	Activated/deactivated by a switch from inside the house called alarm switch	SecurityAlarm =off/on	Activate/Deactivate security alarm switch is pressed	Activate/Deactivate security alarm	SecurityAlarm= on/off
25	Alarm is triggered when the feature is active and a magnetic reed sensor indicates that a window is being opened	SecurityAlarm =on, Alaram=not_set, Windows(_) =closed	Window is opened	Set security alarm	SecurityAlarm= on, Alaram=set, Windows(_) =opened
26	Alarm is triggered when the feature is active and the main door lock sensor indicates that the main door lock is being opened	SecurityAlarm =on, Alaram=not_set, MainDoorLock =closed	MainDoorLock is opened	Set security alarm	SecurityAlarm= on, Alaram=set, MainDoorLock= opened
27	Alarm is triggered when the feature is active and a PIR sensor indicates movement in X1, where X1: Location, X1= {Living room, Bedrooms, Hallway, Kitchen}	SecurityAlarm =on, Alaram=not_set, PIR=negative	Movements	Set security alarm	SecurityAlarm= on, Alaram=set, PIR=positive
28	Alarm is triggered when the feature is active and pressure pads indicate the presence of a person in X2, where X2: location, X2= {Living room, Bedrooms, Hallway}	SecurityAlarm =on, Alaram=not_set, PressurePads =negative	Pressure Pad is pressed	Set security alarm	SecurityAlarm= on, Alaram=set, PressurePads=positive
40	Activated/deactivated by a switch from inside the house called vacation switch	VacationControl=off/on	Activate/Deactivate vacation	Activate/Deactivate	VacationControl =on/off

Username: Administrator | Exclusive edit mode

Figure 9.10: Results of dynamic behaviour attributes identification using IRIS-TS

9.3.2.3 Trigger Events Extraction using IRIS-TS

The Trigger events extraction step is automatically done with no input from the analyst. IRIS-TS will examine the dynamic behaviour attributes identification (Figure 9.10) and automatically extracts all unique trigger events and links them to the requirements they trigger. After that, IRIS-TS will create a new module to correspond for the trigger events extraction table created in step 3 of IRIS (see Section 4.3.4). The created module will contain all unique trigger events and the requirements that each of the trigger events trigger (Figure 9.11).


Formal module '/Smart Homes/TriggerEventExtraction' current 0.0 - DOORS

Standard view | All levels

TriggerEventExtraction
- 1 Activate/Deactivate security a
- 2 Window is opened
- 3 MainDoorLock is opened
- 4 Movements
- 5 Pressure Pad is pressed
- 6 Activate/Deactivate vacation
- 7 Time
- 8 MainDoor is shut
- 9 Unlock main door switch is pre
- 10 Gas/Heat/Smoke sensor is tri
- 11 A/V device is turned on
- 12 Thermostats <> X11
- 13 Increase / decrease of the di
- 14 No movements in X15 for 15
- 15 Night begins
- 16 Day begins
- 17 Water level > 75 %
- 18 Receive a call request, and n
- 19 Humidity sensor is triggered
- 20 Humidity sensor is negative f

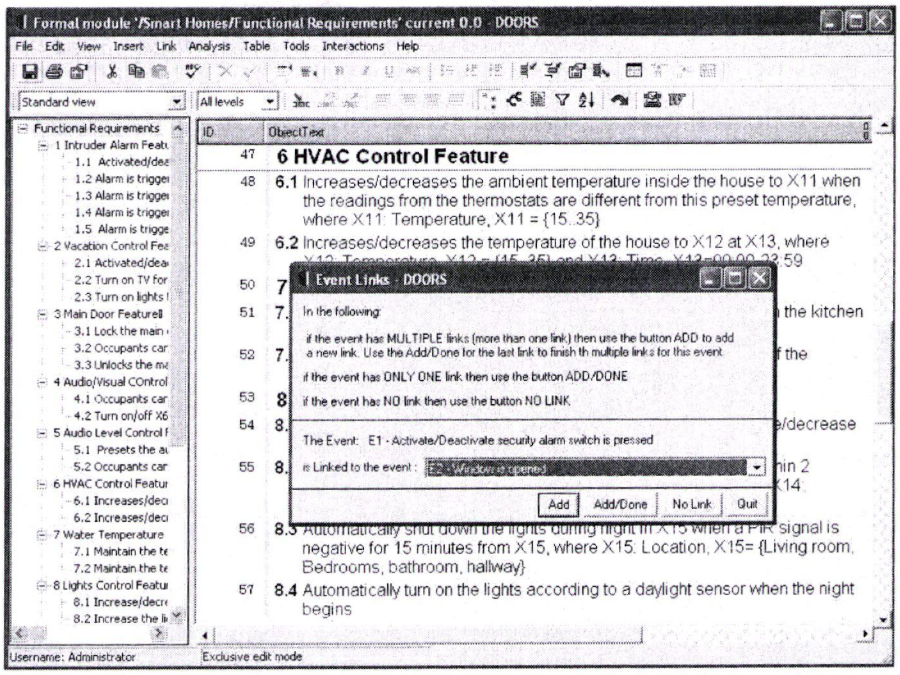
ID	Informal event description	Requirements Triggered by this Event
1	1 Activate/Deactivate security alarm switch is pressed	R24
2	2 Window is opened	R25
3	3 MainDoorLock is opened	R26
4	4 Movements	R27,R55
5	5 Pressure Pad is pressed	R28
6	6 Activate/Deactivate vacation control switch is pressed	R40
7	7 Time	R32,R33,R42,R49,R59,R62, R71
8	8 MainDoor is shut	R35
9	9 Unlock main door switch is pressed	R36
10	10 Gas/Heat/Smoke sensor is triggered	R38,R78
11	11 A/V device is turned on	R44
12	12 Thermostats <> X11	R48
13	13 Increase / decrease of the dimmer slider	R54
14	14 No movements in X15 for 15 minutes	R56
15	15 Night begins	R57
16	16 Day begins	R60
17	17 Water level > 75 %	R64

Username: Administrator | Exclusive edit mode

Figure 9.11: Results of trigger events extraction

9.3.2.4 Linked Events Identification using IRIS-TS

The linked events extraction step is performed by having IRIS-TS examining the trigger events module that was created in the previous step (Figure 9.11). Then, IRIS-TS displays messages to the analyst asking him to determine if the event under investigation is linked to other events. The analyst can choose from a drop down menu that contains all other available events and the analyst can choose as many as he wants. Once IRIS-TS finishes receiving input from the analyst, it will create a module that corresponds to the linked events table created in IRIS step 4 (see section 6.3.5). Figure 9.12 shows the execution of the linked events identification while Figure 9.13 shows the created module by IRIS-TS that corresponds to the linked events identification table of step 4 of IRIS.


Formal module '/Smart Homes/Functional Requirements' current 0.0 - DOORS

ID	ObjectText
47	**6 HVAC Control Feature**
48	**6.1** Increases/decreases the ambient temperature inside the house to X11 when the readings from the thermostats are different from this preset temperature, where X11: Temperature, X11 = {15..35}
49	**6.2** Increases/decreases the temperature of the house to X12 at X13, where
56	**8.3** signal is negative for 15 minutes from X15, where X15: Location, X15= {Living room, Bedrooms, bathroom, hallway}
57	**8.4** Automatically turn on the lights according to a daylight sensor when the night begins

Event Links - DOORS

In the following:

if the event has MULTIPLE links (more than one link) then use the button ADD to add a new link. Use the Add/Done for the last link to finish th multiple links for this event

if the event has ONLY ONE link then use the button ADD/DONE

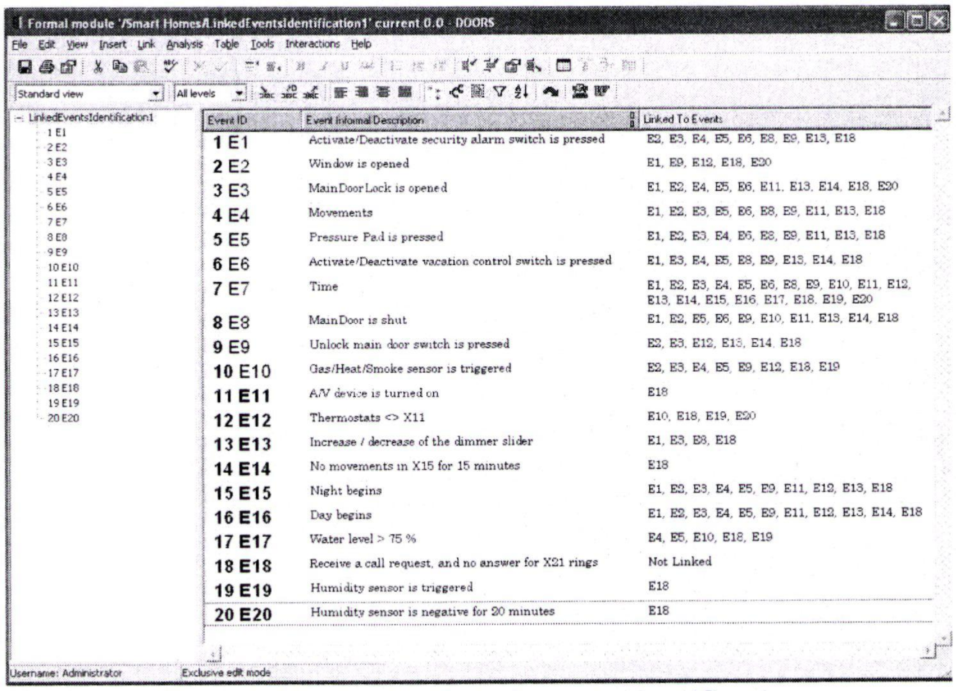
if the event has NO link then use the button NO LINK

The Event: E1 - Activate/Deactivate security alarm switch is pressed

is Linked to the event : E2 - Window is opened

Username: Administrator   Exclusive edit mode

Figure 9.12: Performing linked events identification


Formal module '/Smart Homes/LinkedEventsIdentification1' current 0.0 - DOORS

Event ID	Event Informal Description	Linked To Events
1 E1	Activate/Deactivate security alarm switch is pressed	E2, E3, E4, E5, E6, E8, E9, E13, E18
2 E2	Window is opened	E1, E9, E12, E18, E20
3 E3	MainDoorLock is opened	E1, E2, E4, E5, E6, E11, E13, E14, E18, E20
4 E4	Movements	E1, E2, E3, E5, E6, E8, E9, E11, E13, E18
5 E5	Pressure Pad is pressed	E1, E2, E3, E4, E6, E8, E9, E11, E13, E18
6 E6	Activate/Deactivate vacation control switch is pressed	E1, E3, E4, E5, E8, E9, E13, E14, E18
7 E7	Time	E1, E2, E3, E4, E5, E6, E8, E9, E10, E11, E12, E13, E14, E15, E16, E17, E18, E19, E20
8 E8	MainDoor is shut	E1, E2, E5, E6, E9, E10, E11, E13, E14, E18
9 E9	Unlock main door switch is pressed	E2, E3, E12, E13, E14, E18
10 E10	Gas/Heat/Smoke sensor is triggered	E2, E3, E4, E5, E9, E12, E18, E19
11 E11	A/V device is turned on	E18
12 E12	Thermostats <> X11	E10, E18, E19, E20
13 E13	Increase / decrease of the dimmer slider	E1, E3, E8, E18
14 E14	No movements in X15 for 15 minutes	E18
15 E15	Night begins	E1, E2, E3, E4, E5, E9, E11, E12, E13, E18
16 E16	Day begins	E1, E2, E3, E4, E5, E9, E11, E12, E13, E14, E18
17 E17	Water level > 75 %	E4, E5, E10, E18, E19
18 E18	Receive a call request, and no answer for X21 rings	Not Linked
19 E19	Humidity sensor is triggered	E18
20 E20	Humidity sensor is negative for 20 minutes	E18

Username: Administrator   Exclusive edit mode

Figure 9.13: Results linked events identification

9.3.2.5 Trigger Events Charts Representation using IRIS-TS

The final step in IRIS-TS automatically, without any input from the developer, generates the trigger events charts and saves them in a module called trigger events charts module. Figure 9.14 shows a sample of the generated trigger event charts for event E4. It is worth saying that in Figure 9.14 the button Toggle Length will display the complete text in the diagram or, when repressed, will display a clipped portion of the text to provide uniform non-overlapping display.

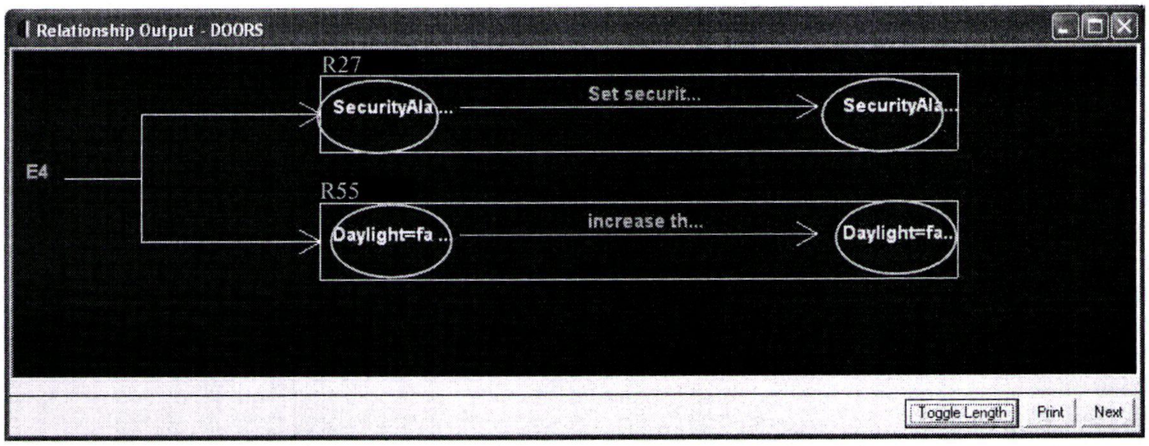


Figure 9.14: Results Trigger Events Charts Representation

9.4 Summary

This chapter presented IRIS-TS which is a tool support that was created as an add-on to DOORS to detect requirements interactions. IRIS-TS was implemented using the DOORS DXL programming language. To show how IRIS-TS works, screenshots are presented from the execution of IRIS-TS on the smart homes policies. These screenshots show that IRIS-TS facilitated the execution of IRIS and provided some automation for the execution of IRIS.

CHAPTER TEN: CONCLUSIONS AND FUTURE WORK

10.1 Summary and Conclusions

Developing software systems has evolved over the years and one of the areas that is considered to be a major key for the success of any new software being developed is requirements engineering. The development of a clear and correct set of stakeholders requirements will heavily contribute to the success of the software being developed. However, in real life, there are always negative relationships and conflicts between requirements which are termed as requirements interaction.

This thesis is devoted to tackle the problem of requirements interactions in software systems. In this regard, a semi-formal approach called IRIS was developed to detect requirements interactions in software systems. IRIS is a semi-formal systematic six step approach that uses tables, graphs, interaction scenarios, and subjective judgment to detect interactions in software systems. IRIS can also be customized by adding plug-ins to its basic core to enhance its performance and make adaptable to any new software domain. IRIS enjoys the advantage of reducing the number of necessary pair-wise comparisons that have to be performed between requirements by discarding irrelevant comparisons that will not lead to interactions. Hence, this can result in a clear reduction in the number of comparisons and consequently reduction in time and effort.

As part of IRIS, a general requirements interaction taxonomy was developed to identify when two requirements are considered interacting. This requirements interaction taxonomy enjoys an in-depth level of details that was lacking in other taxonomies reported in the literature. The requirements interaction taxonomy defines 9 main interaction categories, 24 interaction subcategories, 37 interaction types, and 37 interaction scenarios where each interaction scenario has a corresponding interaction detection guideline that describes how this interaction can be detected.

To validate the proposed IRIS approach, it was applied in three different case studies from different domains. The results obtained by applying IRIS to these case studies have been compared, when possible, to other results reported in the literature. IRIS scores very well compared to other results taking into account that these other approaches used formal methods compared to IRIS which is a semi-formal approach. Although the first two case studies have been exercised by other approaches reported in the literature, the third case study on smart homes can be considered as a main contribution because, to the author's knowledge, no fully documented results for the smart homes case study currently exist in the literature.

Finally, this thesis introduced IRIS-TS, which is a tool support for IRIS that was developed to work within the commercial DOORS requirements management software. IRIS-TS was developed using DOORS DXL which is a special programming language for DOORS that enables users to build their own applications and integrate it in DOORS. The developed code for IRIS-TS would add a separate drop down menu in DOORS main tool bar that enables the user to choose to detect interactions between requirements that are stored in DOORS. IRIS-TS will provide a step by step application of the different steps of IRIS and generate the necessary tables and graphs to facilitate the detection of interaction between requirements.

10.2 Future research

The future research areas should focus on maturing the work done in this thesis and also introducing new ideas that extend the work presented in this thesis.

Future research should focus on the following four areas: Experimental measurement for the effort required to apply IRIS, the development of a framework for interaction detection that combines IRIS with already existing informal and formal approaches, the application of IRIS in new case studies especially in the World Wide Web domain, and finally further development of IRIS-TS. In the following a highlight is given on each of these points.

10.2.1 Experimentation with IRIS

Currently there is a joint research project between the author and fourth year undergraduate students at the Department of Computer Science at the American University in Sharjah. The aim of this project is to provide experimental data regarding the application of IRIS to detect interactions between telephony features. The data will be used to measure IRIS effectiveness.

10.2.2 Development of a three layer framework

The development of a 3-layer framework is one major extension to this thesis. The three layers will be: Informal detection using expert systems, Semi-formal detection using IRIS, and formal detection using a formal language. Aside form studying each layer, the three layers are interconnected to define the potential of cost, effort, and time savings.

10.2.3 Application of IRIS to new Case Studies

The application of IRIS to new case studies is essential to gain more maturity for the approach. The new case studies will help also in designing new plug-ins that can be used by others when applying IRIS in new domains. Finally, new case studies will provide documented interaction results that can be used by developers when developing new software systems to avoid these interactions as early as possible. Currently, there is an interest to apply IRIS in the World Wide Web domain. Future plans include applying IRIS to detect non-functional requirements interactions in the TPC-W Benchmark which is a bench mark used for validating the creation of new E-commerce web sites.

10.2.4 Further Development of IRIS-TS

As it can be seen from Chapter 10, IRIS-TS will execute the first five steps of IRIS and develop the required tables and graphs required to detect interactions in the sixth step of IRIS. However, the tool stops at this point and does not provide any support for the sixth step of IRIS. The reason for that was the reliance on the developer to look at the correct figures and tables and use the interaction scenarios to decide if the two requirements under investigation are interacting. Future plans include the development of a support system to perform the sixth step in DOORS using IRIS-TS. The support will display the correct tables and figures for the two requirements under investigation and choose and display the set of interaction scenarios that are appropriate and can be used with the two requirements being investigated.

Another future improvement of IRIS-TS is to allow the developer to add plug-ins easily through a special window interface and then add these plug-ins automatically to all steps being executed by IRIS-TS.

REFERENCES

[1] J. O. Palmer and N. A. Fields, "An Integrated Environment for Requirements Engineering," *IEEE Software*, vol. 9, pp. 80-85, 1992.

[2] I. Bray, *An introduction to requirements engineering*. Harlow: Addison-Wesley, 2002.

[3] J. A. Goguen and M. Jirotka, *Requirements engineering : social and technical issues*. London: Academic Press, 1994.

[4] E. Hull, K. Jackson, and J. Dick, *Requirements engineering*. London: Springer, 2002.

[5] I. Sommerville and P. Sawyer, *Requirements engineering : a good practice guide*. Chichester, Eng. ; New York: Wiley, 1999.

[6] U. Nikula, J. Sajaniemi, and H. Kalviainen, "A State-of-the-Practice Survey on Requirements Engineering in Small- and Medium-Sized Enterprises," TBRC Research Report 1, Telecom Business Research Center Lappeenranta, Lappeenranta University of Technology. 2000.

[7] C. McPhee and A. Eberlein, "Requirements engineering for time-to-market projects," Ninth Annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, pp. 17-24, 2002.

[8] K. E. Emam and A. Birk, "Validating the ISO/IEC 15504 Measure of Software Requirements Analysis Process Capability," *IEEE Transactions on Software Engineering*, vol. 26, pp. 541-566, 2000.

[9] L. Jiang, A. Eberlein, and B. H. Far, "A Methodology for RE Process Development," 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS), Czech Republic, pp. 263-272, 24-27 May 2004.

[10] M. Shehata, A. Eberlein, and J. Hoover, "Requirements Reuse and Feature Interaction Management," 15th International Conference on Software & Systems Engineering and their Applications (ICSSEA'02), Paris, December 3-5, 2002.

[11] W. N. Robinson, S. D. Pawlowski, and V. Volkov, "Requirements interaction management," *ACM Computing Surveys (CSUR)*, vol. 35, pp. 132-190, June 2003.

[12] N. G. Leveson, *Safeware, System Safety, and Computers*: Addison-Wesley Pub. Co. Inc., 1995.

[13] P. Gibson, "Feature requirements models: Understanding interactions," in *Featutre interactions in telecommunication networks IV*, P. Dini, R. Boutaba, and L. Logrippo, Eds. Amsterdam: IOS press, June 1997, pp. 46-60.

[14] B. Boehm, P. Bose, E. Horowitz, and M. J. Lee, "Software requirements negotiation and renegotiation aids: a theory-W based spiral approach," ICSE-17 Workshop on Formal Methods Application in Software Engineering Practice, pp. 243-253, 1995.

[15] B. Boehm and H. In, "Identifying quality-requirement conflicts," Proceedings of the Second International Conference on Requirements Engineering, Los Alamitos, CA, USA, pp. 218-228, 1996.

[16] J. Ellsberger, A. Sarma, and D. Hogrefe, *SDL : formal object-oriented language for communicating systems*, 2. ed. London: Prentice Hall, 1997.

[17] M. Heisel and J. Souquières., "A heuristic algorithm to detect feature interactions in requirements.," in *Language Constructs for Describing Features*, S. Gilmore and M. Ryan, Eds.: Springer-Verlag London Ltd, 2000/2001, pp. 143-162.

[18] M. Heisel and J. Souquières., "Detecting Feature Interaction - A heuristic approach," First FIREworks Workshop, Germany, pp. 30-48, May 1998.

[19] M. Kolberg, E. Magill, D. Marples, and S. Reiff-Marganiec, "Second Feature Interaction Contest," in *Feature Interactions in Telecommunications and Software Systems*, M. H. Calder and E. H. Magill, Eds.: IOS Press, 2000, pp. 293-310.

[20] E. J. Cameron, N. D. Griffeth, Y.-J. Ling, M. E. Nilson, W. K. Schnure, and H. Velthuijsen, "A feature interaction benchmark for IN and beyond," Proceedings of 2nd International Workshop on Feature Interactions in Telecommunications Software Systems, Amsterdam, Netherlands, pp. 1-23, 1994.

[21] M. Kolberg, E. H. Magill, and M. Wilson, "Compatibility Issues between Services Supporting Networked Appliances," *IEEE Communications Magazine*, vol. 41, pp. 136 - 147, 2003.

[22] S. Reiff-Marganiec and K. J. Turner, "Feature Interaction in Policies," *Computer Networks*, vol. 45, pp. 569-584, March 2004.

[23] Telelogic DOORS, http://www.telelogic.com/products/doorsers/doors/, Last Viewed On May 21, 2005

[24] N. Griffeth, R. Blumenthal, J.-C. Gregoire, and T. Ohta, "Feature Interaction Detection Contest of the Fifth International Workshop on Feature Interactions," *Computer Networks*, vol. 32, pp. 487-510, 2000.

[25] N. Griffeth and Y.-J. Lin, *Feature Interactions in Telecommunications Systems*. St. Petersburg, Florida, USA: IOS Press Inc., 1992.

[26] L. G. Bouma, H. Velthuijsen, and IEEE Communications Society, *Feature interactions in telecommunications systems*. Amsterdam: IOS Press, 1994.

[27] K. E. Cheng and T. Ohta, *Feature Interactions in Telecomunications III*. Kyoto, Japan: IOS Press Inc., 1995.

[28] K. Kimbler and L. G. Bouma, *Feature Interactions in Telecommunications and Software Systems V*. Lund, Sweden: IOS Press, 1998.

[29] M. Calder and E. Magill, *Feature Interactions in Telecommunications and Software Systems VI*. Glasgow, Scotland: IOS Press Inc., 2000.

[30] D. Amyot, *Feature Interactions in Telecommunications and Software Systems VII*. Ottawa, Canada: IOS Press Inc, 2003.

[31] P. Dini, R. Boutaba, and L. Logrippo, *Feature Interactions in Telecommunications Networks*. Montreal, Canada: IOS Press Inc,, 1997.

[32] IEEE Digital Library, http://www.ieee.org, Last Viewed On May 21, 2005

[33] ACM Digital Library, http://www.acm.org, Last Viewed On May 21, 2005

[34] CITESEER Digital Library, http://www.citeseer.com, Last Viewed On May 21, 2005

[35] D. Amyot and L. Logrippo, "Directions in Feature Interaction Research," in *Computer Networks, Volume 45, Issue 5*: ElseVier Sceince Direct, pp. 563-685, August 2004.

[36] M. Calder, M. Kolberg, E. H. Magill, and S. Reiff-Marganiec, "Feature interaction: a critical review and considered forecast," *Computer Networks*, vol. 41, pp. 115-41, 2003.

[37] D. O. Keck and P. J. Kuehn, "The feature and service interaction problem in telecommunications systems: a survey," *IEEE Transactions on Software Engineering*, vol. 24, pp. 779 - 796, October 1998.

[38] Y. Wakahara, M. Fujioka, H. Kikuta, H. Yagi, and S. I. Sakai, "A Method for Detecting Service Interactions," *IEEE Communications*, vol. 31, pp. 32–37, August 1993.

[39] J. Mierop, S. Tax, and R. Janmaat, "Service Interaction in an Object-Oriented Environment," *IEEE Communications*, vol. 31, pp. 46–51, August 1993.

[40] K. Kimbler, E. Kuisch, and J. Muller, "Feature Interaction Among Pan-European Services," in *Feature Interactions in Telecommunications Systems*, L. G. Bouma and H. Velthuijsen, Eds. Amsterdam: IOS Press, May 1994, pp. 73-85.

[41] D. D. Dankel, M. Schmalz, W. Walker, K. Nielsen, L. Muzzi, and D. Rhodes, "An Architecture for Defining Features and Exploring Interactions," in *Feature Interactions in Telecommunications Systems*, L. G. Bouma and H. Velthuijsen, Eds. Amsterdam: IOS Press, May 1994, pp. 258-271.

[42] E. Kuisch, R. Janmaat, H. Mulder, and I. Keesmaat, "A Practical Approach to Service Interactions," *IEEE Communications*, vol. 31, pp. 24-31, August 1993.

[43] D. O. Keck, "A Tool for the Identification of Interaction-Prone Call Scenarios," *Proceedings of the 5th International Workshop on Feature Interactions in Telecommunications and Software Systems*, pp. 276-290, 1998.

[44] K. Kimbler and D. Sobirk, "Use Case Driven Analysis of Feature Interactions," in *Feature Interactions in Telecommunications Systems*, L. G. Bouma and H. Velthuijsen, Eds. Amsterdam: IOS Press, May 1994, pp. 167-177.

[45] I. Sommerville, *Software engineering*, 6th ed. Harlow, England ; New York: Addison-Wesley, 2000.

[46] L. Doldi and L. Doldi, *Validation of Telecom Systems with SDL*: John Wiley & Sons, June 2003.

[47] P. H. J. V. Eijk, C. A. Vissers, and M. Diaz, *The Formal Description Technique Lotos: Results of the Esprit/Sedos Project*: North-Holland, April 1989.

[48] J. D. Hay and J. M. Atlee, "Composing Features and Resolving Interactions," *ACM International Symposium on the Foundations of Software Engineering (FSE)*, pp. 110-119, November 2000.

[49] K. Braithwaite and J. Atlee, "Towards Automated Detection of Feature Interactions," in *Feature Interactions in Telecommunications Systems*: IOS press, 1994, pp. 36-59.

[50] B. Kelly, M. Crowther, J. King, R. Masson, and J. DeLapeyre, "Service Validation and Testing," in *Feature Interactions in Telecommunications Systems III*, K. E. Cheng and T. Ohta, Eds. Amsterdam: IOS Press, October 1995, pp. 173–184.

[51] J. Bredereke, "Families of Formal Requirements in Telephone Switching," in *Feature Interactions in Telecommunication Networks VI*, M. H. Calder and E. H. Magill, Eds. Amsterdam: IOS press, May 2000, pp. 257-273.

[52] P. Zave, "Architectural solutions to feature-interaction problems in telecommunications," Proceedings of 5th International Workshop on Feature Interactions in Telecommunications Software Systems, 29 Sept.-1 Oct. 1998, Lund, Sweden, pp. 10-22, 1998.

[53] P. Zave and M. Jackson, "A Component-Based Approach to Telecommunication Software," *IEEE Software*, vol. 15, pp. 70--78, 1998.

[54] P. Zave and M. Jackson, "New feature interactions in mobile and multimedia telecommunication services," in *Feature Interactions in Telecommunications andSoftware Systems VI*, M. Calder and E. Magill, Eds. Amsterdam: IOS press, May 2000., pp. 51–66.

[55] P. Zave and M. Jackson, "Distributed feature composition: A virtual architecture for telecommunication services," *IEEE Transactions on Software Engineering XXIV*, vol. 10, pp. 831-847, October 1998.

[56] Y. Iraqi and M. Erradi, "An experiment for the processing of feature interactions within an object-oriented environment," in *Feature Interactions in Telecommunication Networks IV*, P. Dini, R. Boutaba, and L. Logrippo, Eds. Amsterdam: IOS Press, June 1997, pp. 298–312.

[57] C. Prehofer, "An object-oriented approach to feature interaction," in *Feature Interactions in Telecommunication Networks IV*, P. Dini, R. Boutaba, and L. Logrippo, Eds. Amsterdam: IOS Press, June 1997, pp. 313–325.

[58] G. Utas, "A pattern language of feature interactions," in *Feature Interactions in Telecommunications and Software Systems V*, K. Kimbler and L. G. Bouma, Eds. Amsterdam: IOS Press, September 1998, pp. 98–114.

[59] L. Blair and J. Pang, "Aspect-Oriented Solutions to Feature Interaction Concerns using AspectJ," in *Feature Interactions in Telecommunications and Software Systems VII*, D. Amyot and L. Logrippo, Eds. Amestrdam: IOS Press, 2003, pp. 87-104.

[60] D. Amyot, L. Charfi, N. Gorse, T. Gray, L. Logrippo, J. Sincennes, B. Stepien, and T. Ware, "Feature Description and Feature Interaction Analysis with Use Case Maps and LOTOS," in *Feature Interactions in Telecommunications and Software Systems VI*, M. Calder and E. Magill, Eds. Amestrdam: IOS Press, 2000, pp. 274-289.

[61] C. Prehofer, "Plug-and-Play Composition of Features and Feature Interactions with Statechart Diagrams," in *Feature Interactions in Telecommunications and Software Systems VII*, D. Amyot and L. Logrippo, Eds. Amsterdam: IOS Press, 2003, pp. 43-58.

[62] K. Berkani, R. Cave, S. Coudert, F. Klay, P. LeGall, F. Ouabdesselam, and J.-L. Richier, "An Environment for Interactive Service Specification," in *Feature Interactions in Telecommunications and Software Systems VII*, D. Amyot and L. Logrippo, Eds. Amestrdam: IOS Press, 2003, pp. 25-41.

[63] A. Metzger and C. Webel, "Feature Interaction Detection in Building Control Systems by Means of a Formal Product Model," in *Feature Interactions in Telecommunications and Software Systems VII*, D. Amyot and L. Logrippo, Eds. Amsterdam: IOS Press, 2003, pp. 105-122.

[64] A. Metzger, "Feature interactions in embedded control systems," *Computer Networks*, vol. 45, pp. 625-44, 2004.

[65] K. J. Turner, "Formalising the Chisel Feature Notation," in *Feature Interactions in Telecommunication Networks VI*, M. H. Calder and E. H. Magill, Eds. Amsterdam: IOS press, May 2000, pp. 241-256.

[66] K. Turner, "Modelling SIP services using CRESS," Formal Techniques for Networked and Distributed Systems (FORTE XV), Berlin, Germany, pp. 162-177, Nov. 2002.

[67] K. J. Turner, "Representing New Voice Services and Their Features," in *Feature Interactions in Telecommunications and Software Systems VII*, D. Amyot and L. Logrippo, Eds. Amsterdam: IOS Press, 2003, pp. 123-140.

[68] S. Reiff-Marganiec and K. J. Turner, "Feature interaction in policies," *Computer Networks*, vol. 45, pp. 569-84, 2004.

[69] P. Zave, H. H. Goguen, and T. M. Smith, "Component coordination: a telecommunication case study," *Computer Networks Journal, Elsevier Science Publishers*, vol. 45, pp. 645-664, 2004.

[70] P. Zave, "Ideal Address Translation: Principles, Properties, and Applications," in *Feature Interactions in Telecommunications and Software Systems VII*, D. Amyot and L. Logrippo, Eds. Amsterdam: IOS Press, 2003, pp. 257-274.

[71] J. H. Choi, H. S. Kim, W. J. Lee, and Y. R. Kwon, "A Petri-Nets Based Approach for Detecting Feature Interactions in Telecommunications Services," 12th Int'l Conference on Computer Communication (ICCC), Seoul, pp. 596–601, August 1995.

[72] J. Bredereke, "Detection of feature interactions in intelligent networks by verification," *Software-Concepts and Tools*, vol. 17, pp. 121-39, 1996.

[73] J. Bredereke, "Formal criteria for feature interactions in telecommunications systems," The IFIP TC6 Conference on Intelligent Networks and New Technologies, London, UK, pp. 68-83, 1996.

[74] C. Klein, C. Prehofer, and B. Rumpe, "Feature Specification and Refinement with State Transition Diagrams," in *Feature Interactions in Telecommunications Networks and Distributed Systems IV*, P. Dini, Ed. Amestrdam: IOS Press, 1997, pp. 284-297.

[75] M. Faci and L. Logrippo, "Specifying features and analysing their interactions in a LOTOS environment," in *Feature Interactions in Telecommuniactions Systems*, L. G. Bouma and H. Velthuijsen, Eds. Amestrdam: IOS Press, 1994, pp. 136-151.

[76] J. Blom, B. Jonsson, and L. Kempe, "Using temporal logic for modular specification of telephone services," Proceedings of 2nd International Workshop on Feature Interactions in Telecommunications Software Systems, pp. 197-213, 1994.

[77] J. P. Gibson, "Towards a feature interaction algebra," Proceedings of 5th International Workshop on Feature Interactions in Telecommunications Software Systems, 29 Sept.-1 Oct. 1998, Lund, Sweden, pp. 217-31, 1998.

[78] P. Gibson, G. Hamilton, and D. Méry, "A taxonomy for triggered interactions using fair object semantics ,," in *Feature Interactions In Telecommunications and Software Systems VI*, M. Calder and E. Magill, Eds. Amestrdam: IOS Press, 2000, pp. 193-210.

[79] A. Felty and K. Namjoshi, "Feature specification and automatic conflict detection," in *Feature Interactions in Telecommunications and Software Systems VI*, M. Calder, E. Magill, Eds. Amsterdam: IOS Press, May 2000, pp. 179-192.

[80] S. M. Rochefort and H. J. Hoover, "An exercise in using constructive proof systems to address feature interactions in telephony," in *Feature Interactions in Telecommunication Networks IV*, P. Dini, R. Boutaba, and L. Logrippo, Eds. AMestrdam: IOS Press, June 1997, pp. 329–341.

[81] M. Frappier, A. Mili, and J. Desharnais, "Detecting feature interactions in relational specifications," in *Feature Interactions in Telecommunication Networks IV*, P. Dini, R. Boutaba, and L. Logrippo, Eds. Amestrdam: IOS Press, June 1997, pp. 123–137.

[82] M. Bostrom and M. Engstedt, "Feature interaction detection and resolution in the Delphi framework," in *Feature Interactions in Telecommunications Systems III*, K. E. Cheng and T. Ohta, Eds. Amestrdam: IOS Press, October 1995, pp. 157–172.

[83] M. Calder and A. Miller, "Generalising Feature Interactions in Email," in *Feature Interactions in Telecommunications and Software Systems VII*, D. Amyot and L. Logrippo, Eds. Amestrdam: IOS Press, 2003, pp. 187-204.

[84] A. Lee, "Formal Specification—A Key to Service Interactions Analysis," Eight Conference on Software Engineering for Telecommunication Systems and Services (SETSS 1992), pp. 62-66, March 1992.

[85] A. Y. H. Lee, "Formal Specification and Analysis of Intelligent Network Services and their Interaction." Australia: Ph.D. Thesis, University of Queensland, December 1992.

[86] M. Butler, "Feature interaction analysis using Z," Technical Report, Broadcom Eireann Research, Dublin, 1993.

[87] R. J. Hall, "Feature combination and interaction detection via foreground/ background models," *Computer Networks Journal, Elsevier Science Publishers*, vol. 32, pp. 449--469, 2000.

[88] M. Plath and M. Ryan, "Defining features for CSP:Reflection on the feature interaction contest," in *Language construct for defining features*, S. Gilmore and M. Ryan, Eds.: Springer verlag, 2000, pp. 202-216.

[89] G. Bruns, P. Mataga, and I. Sutherland, "features as service transformers," in *feature interactions in telecommunications and software systems*, K. Kimbler and W. Bouma, Eds. Amsterdam: IOS press, September 1998, pp. 85-97.

[90] J. Blom, "Formalisation of requirements with emphasis on feature interaction detection," in *Feature Interactions in Telecommunication Networks IV*, P. Dini, R. Boutaba, and L. Logrippo, Eds. Amestrdam: IOS Press, June 1997, pp. 61–77.

[91] P. K. Au and J. M. Atlee, "Evaluation of a sate-based model of feature interactions," in *Feature Interactions in Telecommunication Networks IV*, P. Dini, R. Boutaba, L. Logrippo, Eds. Amestrdam: IOS Press, June 1997, pp. 153–167.

[92] J. Bergstra and W. Bouma, "Models for feature descriptions and interactions," in *Feature Interactions in Telecommunication Networks IV*, P. Dini, R. Boutaba, and L. M. S. Logrippo, Eds. Amestrdam: IOS Press, 1997, pp. 31--45.

[93] T. F. LaPorta, D. Lee, Y.-J. Lin, and M. Yannakakis, "Protocol feature interactions," FORTE-PSTV, pp. 59-74, 1998.

[94] A. Khoumsi, "Detection and resolution of Interactions between Services of the telephone Network," in *Featutre interactions in telecommunication networks IV*, P. Dini, R. Boutaba, and L. Logrippo, Eds. Amestrdam: IOS press, June 1997, pp. 78-92.

[95] A. Khoumsi and R. J. Bevelo, "A detection method developed after a thorough study of the contest held in 1998," in *Feature Interactions in Telecommunications and Software Systems VI*, M. Calder and E. Magill, Eds. Amsterdam: IOS press, May 2000, pp. 229-240.

[96] Y. Inoue, K. Takami, and T. Ohta, "Method for Supporting Detection and Elimination of Feature Interaction in a Telecommunication System," Int'l Workshop Feature Interactions in Telecommunications Software Systems, pp. 61-81, December 1992.

[97] Y. Inoue, K. Takami, and T. Ohta, "Automatic Detection of Service Interactions in Telecommunications Service Specifications," IEEE Int'l Conference on Communications (ICC), New Orleans, pp. 1835-1840, May 1994.

[98] Y. Harada, Y. Hirakawa, and T. Takenaka, "A Design Support Method for Telecommunication Service Interactions," GLOBECOM '91, Phoenix, Ariz., pp. 1661-1666, December 1991.

[99] Y. Harada, Y. Hirakawa, T. Takenaka, and N. Terashima, "A Conflict Detection Support Method for Telecommunication Service Descriptions," *IEICE Trans. Comm.*, vol. E75-B, pp. 986-997, October 1992.

[100] M. Nakamura, Y. Kakuda, and T. Kikuno, "Feature interaction detection using permutation symmetry," in *Feature Interactions in Telecommunications and Software Systems V*, K. Kimbler and L. G. Bouma, Eds. Amestrdam: IOS Press, September 1998, pp. 187–201.

[101] J. G. Thistle, R. P. Malhame, and H.-H. Hoang, "Feature interaction modelling, detection and resolution: A supervisory control approach," in *Feature Interactions in Telecommunication Networks IV*, P. Dini, R. Boutaba, and L. Logrippo, Eds. Amsterdam: IOS Press, June 1997, pp. 93–107.

[102] K. Y. Chan and G. v. Bochmann, "Methods for Designing SIP Features in SDL with Fewer Feature Interactions," in *Feature Interactions in Telecomunications and Software Systems VII*, D. Amyot and L. Logrippo, Eds. Amestrdam: IOS Press, 2003, pp. 59-76.

[103] B. Mitchell, R. Thompson, and C. Jervis, "Phase Automaton for Requirement Scenarios," in *Feature Interactions in Telecommunications and Software Systems VII*, D. Amyot and L. Logrippo, Eds. Amestrdam: IOS Press, 2003, pp. 77-84.

[104] S. Kawauchi and T. Ohta, "Mechanism for 3-way Feature Interactions Occurrence and a Detection System Based on The Mechanism," in *Feature Interactions in Telecommunications and Software Systems*, D. Amyot and L. Logrippo, Eds. Amestrdam: IOS Press, 2003, pp. 313-328.

[105] A. D. Marco and F. Khendek, "eSERL: Feature Interaction Management in Parlay/OSA using Composition Constraints and Configuration Rules," in *Feature Interactions in Telecomunications and Software Systems VII*, D. Amyot and L. Logrippo, Eds. Amestrdam: IOS Press, 2003, pp. 247-254.

[106] I. Aggoun and P. Combes, "Observers in the SCE and the SEE to Detect and Resolve Service Interactions," in *Feature Interactions in Telecommunication Networks IV*, P. Dini, R. Boutaba, and L. Logrippo, Eds. Amsterdam: IOS Press, June 1997, pp. 198-212.

[107] F. J. Lin and Y. J. Lin, "A Building Block Approach to Detecting and Resolving Feature Interactions," in *Feature Interactions in Telecommunications Systems*, L. G. Bouma and H. Velthuijsen, Eds. Amsterdam: IOS Press, May 1994, pp. 186-191.

[108] M. Nakamura, P. Leelaprute, K. Matsumoto, and T. Kikuno, "Detecting Script-to-Script Interactions in Call Processing Language," in *Feature Interactions in Telecommunications and Software Systems VII*, D. Amyot, Ed. Amestrdam: IOS Press, 2003, pp. 215-230.

[109] M. Nakamura, P. Leelaprute, K.-i. Matsumoto, and T. Kikuno, "On detecting feature interactions in the programmable service environment of Internet telephony," *Computer Networks*, vol. 45, pp. 605-624, 2004.

[110] P. Combes and S. Pickin, "Formalisation of a user view of network and services for feature interaction detection," Proceedings of 2nd International Workshop on Feature Interactions in Telecommunications Software Systems, Amsterdam, Netherlands, pp. 120-35, 1994.

[111] M. Plath and M. Ryan, "Plug-and-play features," Proceedings of 5th International Workshop on Feature Interactions in Telecommunications Software Systems, Amsterdam, Netherlands, Lund, Sweden, pp. 150-64, 1998.

[112] M. Calder and A. Miller, "Using SPIN for feature interaction analysis - a case study," Model Checking Software. 8th International SPIN Workshop, 19-20 May 2001, Toronto, Ont., Canada, pp. 143-62, 2001.

[113] B. Stepien and L. Logrippo, "Representing and verifying intentions in telephony features using abstract data types," in *Feature Interactions in Telecommunications Systems III*, K. E. Cheng and T. Ohta, Eds. Amestrdam: IOS Press, October 1995, pp. 141–155.

[114] C. Capellmann, P. Combes, J. Petterson, B. Renard, and J. L. Ruiz, "Consistent interaction detection – a comprehensive approach integrated with service creation," in *Feature Interactions in Telecommunication Networks IV*, P. Dini, R. Boutaba, and L. Logrippo, Eds. Amestrdam: IOS Press, June 1997, pp. 183–197.

[115] J. Kamoun and L. Logrippo, "Goal-oriented feature interaction detection in the intelligent network model," in *Feature Interactions in Telecommunications and Software Systems V*, K. Kimbler and L. G. Bouma, Eds. Amestrdam: IOS Press, September 1998, pp. 172–186.

[116] L. Du Bousquet, F. Ouabdesselam, J.-L. Richier, and N. Zuanon, "Feature interaction detection using a synchronous approach and testing," *Computer Networks*, vol. 32, pp. 419-31, 2000.

[117] L. d. Bousquet, F. Ouabdesselam, J.-L. Richier, and N. Zuanon, "Incremental feature validation: a synchronous point of view," in *Feature Interactions in Telecommunications and Software Systems V*, K. Kimbler and L. G. Bouma, Eds. Amestrdam: IOS Press, Septmber 1998, pp. 262–275.

[118] D. P. Guelev, M. D. Ryan, and P. Y. Schobbens, "Feature Integration as Substitution," in *Feature Interactions in Telecommunications and Software Systems*, D. Amyot and L. Logrippo, Eds. Amsterdam: IOS Press, 2003, pp. 275-294.

[119] M. Thomas, "Modelling and analysing user views of telecommunications services," in *Feature Interactions in Telecommunication Networks IV*, P. Dini, R. Boutaba, and L. Logrippo, Eds. Amestrdam: IOS Press, June 1997, pp. 168–182.

[120] W. Bouma, W. Levelt, A. Melisse, K. Middelburg, and L. Verhaard, "Formalization of Properties for Feature Interaction Detection: Experiece in a Real-Life Situation," Second Int'l Conference on Intelligence in Broadband Services and Networks, pp. 393-405, Septmber 1994.

[121] A. Gammelgaard and J. E. Kristensen, "Interaction Detection, A Logical Approach," in *Feature Interactions in Telecommunications Systems*, L. G. Bouma and H. Velthuijsen, Eds. Amestrdam: IOS Press, May 1994, pp. 178-196.

[122] P. Ladkin, "The Risks Digest," *ACM SIGSOFT Software Engineering Notes*, vol. 15, 1995.

[123] J. Mylopoulos, L. Chung, and B. Nixon, "Representing and Using Non-Functional Requirements: A process-Oriented Approach," *IEEE Transactions on Software Engineering, Special Issue on Knowledge Representation and Reasoning in Software Development*, vol. 18, pp. 483-497., June 1992.

[124] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke, "Viewpoints: a framework for integrating multiple perspectives in system development," *International Journal of Software Engineering and Knowledge Engineering*, vol. 2, pp. 31-58, 1992.

[125] J. Mylopoulos, L. Chung, and B. Nixon, "Representing and using nonfunctional requirements: a process-oriented approach," *IEEE Transactions on Software Engineering*, vol. 18, pp. 483-497, 1992.

[126] J. J. Poorman, *Data flow diagrams: Unraveling the mystery*: The Project Office, 1988.

[127] R. Grosu, C. Klein, B. Rumpe, and M. Broy, "State Transition Diagrams," *TUM-I9630*, 1996.

[128] A. V. Lamsweerde and E. Letier, "Handling obstacles in goal-oriented requirements engineering," *IEEE Trans. Software Engineering*, vol. 26, pp. 978–1005, 2000.

[129] J. Beck, *A survey of program slicing for software engineering*: Research Institute for Computing and Information Systems, University of Houston--Clear Lake National Aeronautics and Space Administration National Technical Information Service, 1993.

[130] S. Fickas, "A knowledge-based approach to specification acquisition and construction," CIS-TR-85-13. University of Oregon,, Eugene, OR. 1985.

[131] S. Fickas and J. Anderson, "A proposed perspective shift: Viewing specification design as a planning problem," Fifth International Workshop on Software Specification and Design, Los Alamitos, CA, pp. 177–184, 1989.

[132] W. N. Robinson, "Automated negotiated design integration: Formal representations and algorithms for collaborative design," University of Oregon, Eugene, OR. 1993.

[133] K. L. Heninger, "Specifying software requirements for complex systems: New techniques and their application," *IEEE Trans. Software Engineering*, pp. 2-13, 1980.

[134] E. Cameron, "A Feature Interaction Benchmark for IN and Beyond," *E.J. Cameron et al., A Feature Interaction Benchmark for IN and Beyond, in Feature Interactions in Telecommunications Systems, IOS press, pp. 1-23, 1994.*

[135] P. Gibson, G. Hamilton, and D. Mery, "A Taxonomy for Triggered Interactions using Fair Object Semantics," in *Feature Interactions in Telecommunications and Software Systems*, E. Magill, Ed.: IOS Press, 2000, pp. 193-209.

[136] N. Gorse, "The Feature Interaction Problem: Automated filtering of Incoherences & Generation of Validatation Test suites at the Design Stage," University of Ottawa, Ottawa,Ontario,Canada, 2001.

[137] M. Frappier, A. Mili, and J. Desharnais, "Defining and detecting feature interactions," Algorithmic Languages and Calculi, pp. 212-239, 1997.

[138] M. Shehata, A. Eberlein, and A. O. Fapojuwo, "Feature Interactions between Networked Smart Home Appliances," QSSE, 4th ASERC Workshop on Quantitative and Soft Computing Based Software Engineering, Banff, Alberta, Canada, pp. 50-54, February 16-17 2004.

[139] M. Shehata and A. Eberlein, "Requirements interaction detection using semi-formal methods," Proceedings 10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems. ECBS 2003, 7-10 April 2003, Huntsville, AL, USA, pp. 224-232, 2003.

[140] T. F. Bown, "The Feature Interaction Problem in Telecommunication Systems," the 7th SETS Conference, pp. 59-62, 1989.

[141] R. J. Hall, "Submission to the Second feature interaction contest," Technical report, AT&T Labs Research 2000.

[142] D. Samborski, "Submission to the second feature interaction contest," technical report, Loria labs 2000.

[143] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*: Addison-Wesley Professional, 2004.

[144] T. Pender, *UML Bible*: Wiley Inc., 2003.

[145] M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*: Addison-Wesley Professional, 2003.

[146] S. R. Schach, *Object-oriented and classical software engineering*, 5th ed. Boston: McGraw-Hill, 2001.

[147] D. Amyot, "Use Case Maps as a Feature Description Notation," in *FIREworks Feature Constructs Workshop*. Glasgow, Scotland, UK, May 2000, pp. 27-44.

[148] D. Amyot, R. Buhr, T. Gray, and L. Logrippo, "Use case maps for the capture and validation of distributed system requirements," *RE'99: Fourth IEEE International Symposium on Requirements Engineering, Ireland*, pp. 44-53, June 1999.

[149] R. J. A. Buhr and R. S. Casselman, *Use Case Maps for Object-Oriented Systems*: Prentice Hall, 1995.

[150] M. Heisel and J. Souquières., "A heuristic algorithm to detect feature interactions in requirements.," in *Language Constructs for Describing Features*, M. Ryan, Ed.: Springer-Verlag London Ltd, 2001, pp. 143-162.

[151] L. Harte and R. Flood, *Introduction to Public Switched Telephone Networks (PSTN), Local Loop, Switching, DSL, ATM, SS7, and AIN*: Althos Publishing, 2003.

[152] M. Plath and M. D. Ryan, "Entry for FIW'00 Feature Interaction Contest," Technical Report, University of Birmingham 2000.

[153] M. Nakamura, T. Ding, J. Sincennes, X. Lu, and L. Logrippo, "Submission to the second feature interaction contest," Technical report, University of Ottawa 2000.

[154] H. L. Lutfiyya, J. Moffett, and F. Garcia, *Proceedings of 4th IEEE International Workshop on Policies for Distributed Systems and Networks*: IEEE Computer Society, Italy, 2003.

[155] J. B. Michael, J. Lobo, and N. Dulay, *Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks*: IEEE Computer Society, Los Alamitos, California, USA, 2002.

[156] M. Sloman, J. Lobo, and E. C. Lupu, *Proceedings of the 2nd international workshop on Policies for Distributed Systems and Networks*: Springer Verlag Lecture Notes in Computer Science, 2001.

[157] D. Verma, M. Devarakonda, E. Lupu, and M. Kohli, *Proceedings of the 5th IEEE International Workshop on Policies for Distributed Systems and Networks*: IEEE Computer Society, New York, USA, 2004.

[158] M. Amer, A. Karmouch, T. Gray, and S. Mankovskii, "Feature interaction resolution using fuzzy policies," in *Feature Interactions in Telecommunications and Software Systems VI*, M. Calder and E. E. Magill, Eds. Amsterdam: IOS Press, Inc., 2000, pp. 94-112.

[159] E. C. Lupu and M. Sloman, "Conflicts in policy-based distributed systems management," *IEEE Transactions on Software Engineering*, vol. 25(6), pp. 852-869, 1999.

[160] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "Ponder: A language specifying security and managements policies for distributed systems," Technical Report, Imperial College, London 2000.

[161] G. Yee and L. Korba, "Feature interactions in policy driven management," in *Feature Interactions in Telecommunications and Software Systems VII*, D. Amyot and L. Logrippo, Eds.: IOS Press, 2003, pp. 231-238.

[162] A. D. Marco and F. Khendek, "eSERL: Feature interaction in Parlay/OSA using composition constraints and configuration rules," in *Feature Interactions in Telecommunications and Software Systems VII*, D. Amyot and L. Logrippo, Eds. Amsterdam: IOS Press, June 2003, pp. 247-254.

[163] S. Reiff-Marganiec and K. J. Turner, "A policy architecture for enhancing and controlling features," in *Feature Interactions in Telecommunications and Software Systems VII*, D. Amyot and L. Logrippo, Eds. Amsterdam: IOS Press, 2003, pp. 239-246.

[164] O. Hersent, J.-P. Petit, and D. Gurle, *Beyond VoIP Protocols: Understanding Voice Technology and Networking Techniques for IP Telephony*: John Wiley & Sons Canada, Ltd, 2005.

[165] M. Edge, B. Taylor, G. Dewsbury, and M. Groves, "The Potential for 'Smart Home' systems in meeting the care needs of older persons and people with disabilities," *Seniors' Housing Update*, vol. 10, pp. 6-7, 2000.

[166] J. M. M. Ferreira, T. Amaral, D. Santos, A. Agiannidis, and M. Edge, "The Custodian Tool: Simple Design of Home Automation Systems for People with Special Needs," EIB Scientific Conference, Munich, Germany, 2000.

[167] R. L. Smith, *Smart House: The coming revolution in housing*: GP Publishing Inc, 1988.

[168] D. Briere and P. Hurley, *Smart homes for dummies*: 2nd edition, Wiley Publishing Inc, 2003.

APPENDIX A: PUBLICATIONS

1. Mohamed Shehata, Armin Eberlein, Abraham Fapojuwo, "Managing Policy Interactions in KNX based Smart Homes", Submitted for Publication, International Journal on Network and Computer Application, Elsevier Pub. Co.
2. Mohamed Shehata, Armin Eberlein, Abraham Fapojuwo, "Using Semi-Formal Methods For Detecting Interactions Among Smart Homes Policies", in Preparation for submission to the International Journal of Computer Networks, Elsevier Pub. Co.
3. Mohamed Shehata, Armin Eberlein, Abraham Fapojuwo, AbdAllah Mohamed, "A Taxonomy for Identifying Requirements Interactions in Software Systems", in Preparation for submission to the International Journal of Requirements Engineering, Elsevier Pub. Co.
4. AbdAllah Mohamed, Gunther Ruhe, Armin Eberlein, Mohamed Shehata, "A Basis for Managing Attributed Objects Inconsistencies", in Preparation for submission to the International Journal of Information and Software Technology, Elsevier Pub. Co.
5. Mohamed Shehata, Armin Eberlein, Abraham Fapojuwo, " Investigating the Problem of Feature Interactions in Product Families", 2nd Annual Engineering Graduate Students Conference, pp. 71,May 2-3 2005, University of Calgary, Calgary, Alberta, Canada.
6. Mohamed Shehata, Armin Eberlein, Abraham Fapojuwo, "IRIS: A Semi Formal Approach for Detecting Requirements Interactions", ECBS 2004, 11th IEEE

International Conference and Workshop on the Engineering of Computer Based Systems, pp. 273-281, May 24-27 2004, Brno, Czech Republic

7. Mohamed Shehata, Li Jiang, Armin Eberlein, "Requirements Interaction Detection Process Guide", CCECE 2004, IEEE Canadian Conference on Electrical and Computer Engineering, pp. 1753-1756, May 2-5, 2004, Niagara Falls, Ontario, Canada.
8. Mohamed Shehata, Armin Eberlein, Abraham Fapojuwo, "The Use of Semi-Formal Methods for Detecting Requirements Interactions", SE 2004, The IASTED International Conference on Software Engineering, pp.230-235, February 17-19 2004, Innsbruck, Austria
9. Mohamed Shehata, Armin Eberlein, Abraham Fapojuwo, "Feature Interactions between Networked Smart Home Appliances", QSSE 2004, 4th ASERC Workshop on Quantitative and Soft Computing Based Software Engineering, pp. 50-54, February 16-17 2004, Banff, Alberta, Canada
10. Mohamed Shehata, Armin Eberlein, Abraham Fapojuwo, "Detecting Requirements Interactions: A Three-Level Framework", ASE 2003, Proceedings of the 18th IEEE Conference on Automated Software Engineering, pp. 352-355, October 6-10, 2003, Montreal, Canada
11. Mohamed Shehata, Armin Eberlein, "Requirements Interaction Detection using Semi-Formal Methods", ECBS 2003, 10th IEEE Symposium and Workshops on Engineering of Computer Based Systems Huntsville Alabama, pp. 224-232, USA April 7-11, 2003

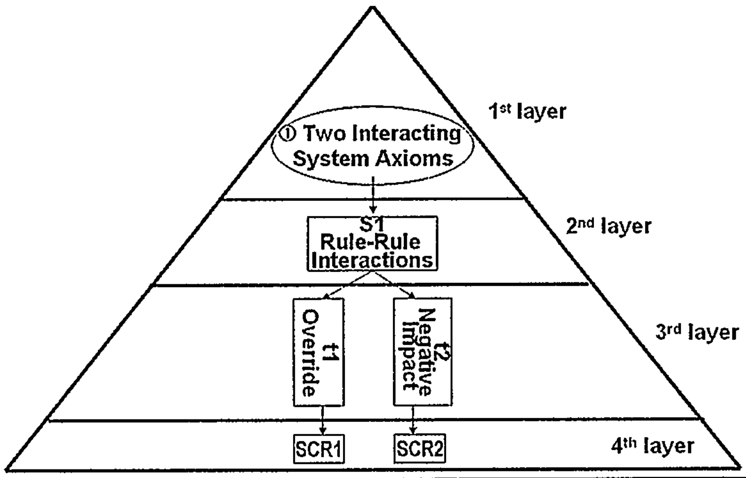
12. Mohamed Shehata, Armin Eberlein, "Issues in Requirements Reuse and Feature Interaction Management", ICSSEA 2002 - 15th International Conference on Software Systems Engineering and their Applications. Paris, France, Vol. 1, No. 3-5, pp. 1-7, December 3-5, 2002

13. Mohamed Shehata, Armin Eberlein, "Requirements Interaction Management: A Multi-Level Framework", SEA 2002 - The 6th IASTED International Conference on Software Engineering and Applications. MIT, USA, pp. 88-93, November 4-6, 2002

APPENDIX B: FULL RESULTS FROM CHAPTER 3 ON THE REQUIREMENTS INTERACTION TAXONOMY

This appendix presents the rest of the proposed interaction taxonomy presented in Chapter 3. The examples used in each set of interaction scenarios are taken from the same domain to provide more understanding and consistency within a single domain.

B.1. Two Interacting System Axioms[1]

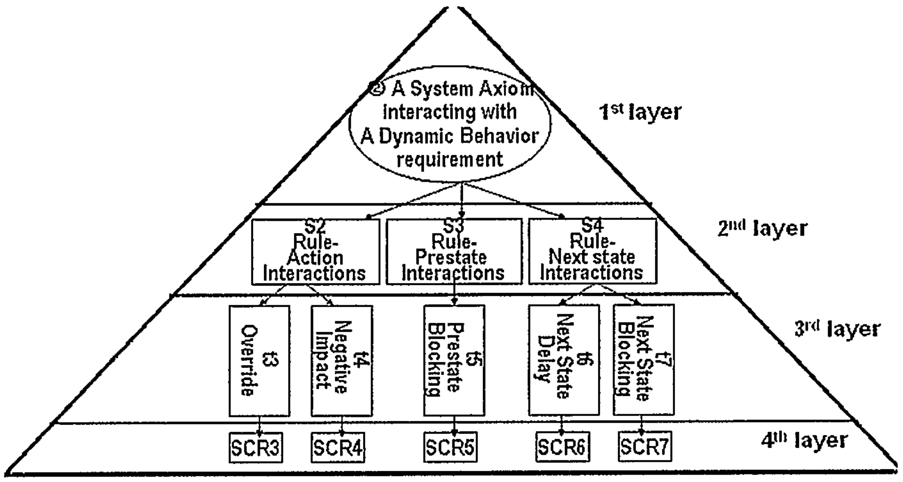


Scenario ID	SCR1
Type of Interaction	TwoInteractingSystemAxioms → Rule-RuleInteractions → Override
Detection Guideline	IF {(R1.Rule OVERRIDES R2.Rule)} THEN {R1 is interacting with R2 under the t1 interaction type}
Example	• R1(Security) "The library page on the website shall be always be under secure logon for members only using (X=username/password) technique" • R2(Usability) "All website pages are accessible by no more than 2 clicks from the menu bar" • Interaction: What happen if a user, who is not signed in, wants to go to the library page? In this case the security requirement R1 overrides R2 and redirects him to a sign in page. This means that the user, assuming he is a member, needs several clicks to go to the library page.
Parameters Effect	If X is changed to automatic IP detection and the secure logon is granted to specific IPs then the interaction is eliminated as the user who tries to go to the library page is validated using his IP address. Of course if the user tries to access the library from an unknown machine then he has to go through the username/password validation.

Scenario ID	SCR2
Type of Interaction	TwoInteractingSystemAxioms → Rule-RuleInteractions → NegativeImpact
Detection Guideline	IF {(R1.Rule NEGATIVELY_IMPACTS R2.Rule)} THEN {R1 is interacting with R2 under the t2 interaction type}
Example	• R1(Assurance) "There shall be an input acceptability checking mechanism X to validate the input data before the system exhibits any response" • R2(Performance) "The response time of the system should be as minimal as possible and at all times should be equal to (Y=0 – 3.0 seconds)" • Interaction: What happens if the input acceptability mechanism X is set to a very complex mechanism? This will cause the system response time to increase dramatically which negatively impacts R2.
Parameters Effect	If the input acceptability mechanism X is set to a simple mechanism then the system response time is reduced and the negative impact is small or can be neglected.

[1] The examples of interaction scenarios in this category are taken from the web e-commerce system which is a representative of the web domain

B.2. A System Axiom Interacting with a Dynamic Behaviour Requirement [2]


A System Axiom Interacting with A Dynamic Behavior requirement
1st layer
S2 Rule-Action Interactions
S3 Rule-Prestate Interactions
S4 Rule-Next state Interactions
2nd layer
t3 Override
t4 Negative Impact
t5 Prestate Blocking
t6 Next State Delay
t7 Next State Blocking
3rd layer
SCR3
SCR4
SCR5
SCR6
SCR7
4th layer

Scenario ID	SCR3
Type of Interaction	SystemAxiomInteractingWithDynamicBehaviourRequirement → Rule-ActionInteractions → Override
Detection Guideline	IF {(R1.Rule OVERRIDES R2.Action)} THEN {R1 is interacting with R2 under the t3 interaction type}
Example	• R1: "The max temperature of hot water from boiler is 45 degrees in order to keep the boiler in safe operation" " • R2: Increase the temperature of the hot water to (X=55) degrees in outlet (Y=washing machine) when the washing machine starts operating" • Interaction: Obviously the Rule of R1 will override the action of R2 and will not allow the increase of the temperature to 55 degrees
Parameters Effect	If X is changed to be a less or equal to 45 degrees then the interaction is eliminated

Scenario ID	SCR4
Type of Interaction	SystemAxiomInteractingWithDynamicBehaviourRequirement → Rule-ActionInteractions → NegativeImpact
Detection Guideline	IF {(R1.Action NEGATIVELY_IMPACTS R2.Rule)} THEN {R1 is interacting with R2 under the t4 interaction type}
Example	• R1 "Executive floor calls are of highest priority" • R2 "The lift is called by pressing the call button and it should arrive within (X=2min) minutes otherwise an alternative car is assigned to that floor" • Interaction: If there are calls from the executive floor then the arrival of the lift is delayed until executive floor calls are served. Hence the rule of R1 has negatively affected the action of R2 by delaying the arrival of the lift
Parameters Effect	If X is changed to longer wait period then the interaction can be reduced

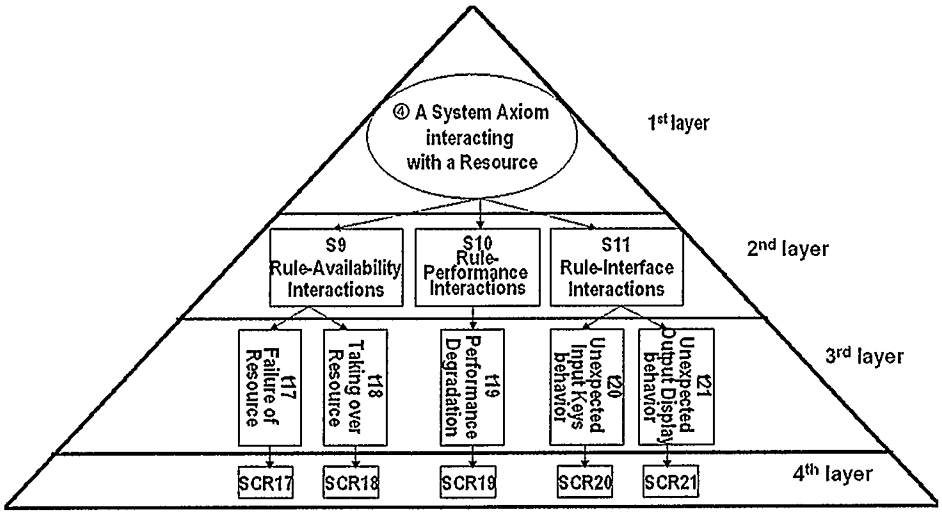
Scenario ID	SCR5
Type of Interaction	SystemAxiomInteractingWithDynamicBehaviourRequirement → Rule-PreStateInteractions → PreStateBlocking
Detection Guideline	IF {(R1.Rule BLOCKS R2.PreState)} THEN {R1 is interacting with R2 under the t5 interaction type }
Example	• R1(maintenance) "To avoid system problems, the lift is subjected to regular maintenance on monthly bases " • R2(operation) "When lift is on standby at floor (X=K) with doors closed and receives call from floor K, it opens its doors" • Interaction: what happens when there is a maintenance going on with the lift at floor K and someone calls the lift from floor K? Obviously it will not open its doors because the power is disconnected during maintenance to prevent accidents. Hence the rule of R1 has prevented and blocked the lift from being in standby which is prestate of R2
Parameters Effect	If X has changed to be the parking level of the lift, then the user will not have access to call the lift from this level. Of course he can still call lift from other levels but in such case the interaction is not prestate blocking between R1 and R2.

[2] The examples of interaction scenarios in this category are taken from the lift system which is a representative of the control domain

Scenario ID	SCR6
Type of Interaction	SystemAxiomInteractingWithDynamicBehaviourRequirement → Rule-NextStateInteractions → NextStateDelay
Detection Guideline	IF {(R1.Rule DELAYS R2.NextState)} THEN {R1 is interacting with R2 under the t6 interaction type }
Example	• R1 (Operation) "Executive floor calls always has (X=highest priority)" • R2 (Operation) "When the lift passes by floor K and there is a call from this floor, the lift will stop at floor K" • Interaction: What happens when the lift is passing by floor K and there is a call from floor K but there is always 5 calls from executive floors. In this case, R2 next state will not be reached which is to stop at floor K until all executive calls are served. Hence the rule of R1 has delayed the next state of R2.
Parameters Effect	What happens when there is continues calls from executive floors? This means that the lift won't go to floor K which means that there is a severe delay to go next state of R2. But if X is changed to be highest priorities for 5 calls then the lift must serve regular floors then the severity of the interaction is reduced.

Scenario ID	SCR7
Type of Interaction	SystemAxiomInteractingWithDynamicBehaviourRequirement → Rule-NextStateInteractions → NextStateBlocking
Detection Guideline	IF {(R1.Rule BLOCKS R2.NextState)} THEN {R1 is interacting with R2 under the t7 interaction type }
Example	• R1 "for a lift at floor K, The lift doors eventually must close after a maximum of (x=1 minute)" • R2 "When something blocks lift doors, the lift interrupts the process of closing the doors and reopens them" • Interaction: if a user keeps blocking the lift doors with his leg then after a 1 minute the rule of R1 is enforced and prevents R2 from being able to reach its next state which is "Doors opened"
Parameters Effect	If X is changed to be 1 hour or unlimited time then there is no interaction

B.3. A System Axiom Interacting with a Resource[3]


④ A System Axiom interacting with a Resource
1st layer
S9 Rule-Availability Interactions
S10 Rule-Performance Interactions
S11 Rule-Interface Interactions
2nd layer
t17 Failure of Resource
t18 Taking over Resource
t19 Performance Degradation
t20 Unexpected Input Keys behavior
t21 Unexpected Output Display behavior
3rd layer
SCR17
SCR18
SCR19
SCR20
SCR21
4th layer

Scenario ID	SCR17
Type of Interaction	SystemAxiomInteractingWithResource → Rule-AvailabilityInteractions → FailureOfResource
Detection Guideline	IF {(R1.Rule violates Resource.Availability) AND (R1.Rule LEADS_TO_FAILURE Resource.Availability)} THEN {R1 is interacting with Resource under t17 interaction type }
Example	• R1 "The website shall be able to handle (X=5 hits/sec)" • Resource.Availability "The application server must be available for processing requests more than 99.9% during each week" • Interaction: If the website receives heavy load, say 100 hits/sec. at a single instance then this might cause a failure to the application server. This is due to the fact that the website was not designed to receive such amount of request. If this occurred frequently then the rule of R1 has caused failure rate of application server to exceed its constraint stated in resource availability.
Parameters Effect	If X is increased to a reasonable number then the interaction is eliminated.

Scenario ID	SCR18
Type of Interaction	SystemAxiomInteractingWithResource → Rule-AvailabilityInteractions → TakingOverResource
Detection Guideline	IF {(R1.Rule violates Resource.Availability) AND (R1.Rule LEADS_TO_TAKING_OVER Resource.Availability)} THEN {R1 is interacting with Resource under t18 interaction type }
Example	• R1:"The system shall use X database server to store and retrieve data" • Resource.Availability "The database server must be available for processing requests more than 99.9% during each week" • Interaction: Assuming that the database server is an old one that can handle only few requests simultaneously. Every time an application server sends a few requests to database server, the database server gets busy and can't handle new requests. If there is more than one application server accessing this database server then it is often unavailable for other application servers.
Parameters Effect	If the database server X is set to a new and powerful one then it becomes more available to all application servers and won't appear as being taken over by just one application server

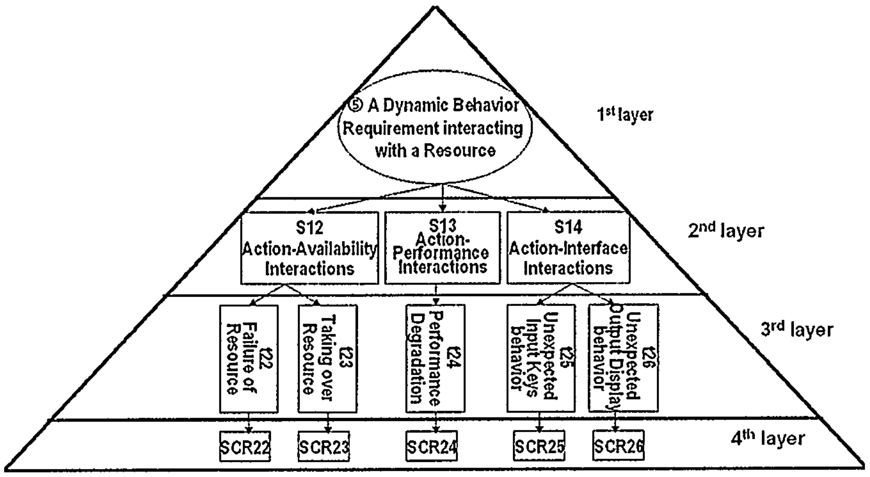
[3] The examples of interaction scenarios in this category are taken from the web e-commerce system which is a representative of the web domain

Scenario ID	SCR19
Type of Interaction	SystemAxiomInteractingWithResource → Rule-PerformanceInteractions → PerformanceDegradation
Detection Guideline	IF {(R1.Rule violates Resource.Performance) AND (R1.Rule LEADS_TO_PERFORMANCE_DEGRADATION Resource.Performance)} THEN {R1 is interacting with Resource under t19 interaction type }
Example	• R1:"The system shall use X techniques for encryption of transmitted financial data " • Resource.Performance "The response time of the application server is less than 3 seconds " • Interaction: Assume that X is a very complex technique. Every time a user tries to submit financial data, the server must encrypt the data using X and since X is very complex then its performance is degraded for any other requests and it can also exceed the 3 seconds limit in the constraint of server performance
Parameters Effect	If X is set to a normal encryption technique then the interaction is eliminated

Scenario ID	SCR20
Type of Interaction	SystemAxiomInteractingWithResource → Rule-InterfaceInteractions → UnexpectedInputKeysBehabvior
Detection Guideline	IF {(R1.Rule violates Resource.Interface) AND (R1.Rule LEADS_TO_UNEXPECTED_INPUT_ BEHAVIOUR Resource.Interface)} THEN {R1 is interacting with Resource under t20 interaction type }
Example	• R1:"User can accept incoming calls on the net phone using (X= pressing number 9 number key, which is letter Y to stand for YES) technique" • Resource.Interface "Standard input interface is provided for the net phone interface" • Interaction: If there is an incoming call and the user is not familiar with this technique, the user might press the regular keys for accepting new calls but it won't work so he might try different keys which might result for terminating the incoming call unexpectedly
Parameters Effect	Setting X to only standard techniques eliminate the interaction

Scenario ID	SCR21
Type of Interaction	SystemAxiomInteractingWithResource → Rule-InterfaceInteractions → UnexpectedOutputDisplayBehabvior
Detection Guideline	IF {(R1.Rule violates Resource.Interface) AND (R1.Rule LEADS_TO_ UNEXPECTED_OUTPUT_ DISPLAY Resource.Interface)} THEN {R1 is interacting with Resource under t21 interaction type }
Example	• R1:"The user is notified by incoming calls on his net phone using (X= switch the focus to the net phone incoming message interface and keeps the user there until he provides a response) technique" • Resource.Interface "Standard output interface is provided for the net phone interface" • Interaction: If the user is playing a game on the screen and there is an incoming call then the focus is switched automatically to the net phone and the user loses the game he is playing (which in sometimes might be more important than the incoming call) which results in unexpected display behaviour.
Parameters Effect	If X is set to sounding an alarm with a background visual alarm then there is no interaction as the user will not be surprised by an unusual display behaviour

B.4. A Dynamic Behaviour Requirement Interacting with a Resource[4]


⑤ A Dynamic Behavior Requirement interacting with a Resource
1st layer
S12 Action-Availability Interactions
S13 Action-Performance Interactions
S14 Action-Interface Interactions
2nd layer
t22 Failure of Resource
t23 Taking over Resource
t24 Performance Degradation
t25 Unexpected Input Keys behavior
t26 Unexpected Output Display behavior
3rd layer
SCR22
SCR23
SCR24
SCR25
SCR26
4th layer

Scenario ID	SCR22
Type of Interaction	DynamicBehaviourRequirementInteractingWithResource → Action-AvailabilityInteractions → FailureOfResource
Detection Guideline	IF {(R1.Action violates Resource.Availability) AND (R1.Action LEADS_TO_FAILURE Resource.Availability)} THEN {R1 is interacting with Resource under t22 interaction type }
Example	• R1 "When the electricity consumption exceeds X KW/hr, start shutting down devices A then B then C then D in this order until consumption reaches Y KW/hr" • Resource.Availability "The boiler shall be available more than 99.9% during each week" • Interaction: Assume that the boiler is device C in R1. If X is set to a small number then it is often that the system shall shutdown the boiler to maintain the consumption rate and this will violate the resource availability constraint.
Parameters Effect	If X is set to a large number or the boiler is not in the list of devices to be shutdown then there is no interaction

Scenario ID	SCR23
Type of Interaction	DynamicBehaviourRequirementInteractingWithResource → Action-AvailabilityInteractions → TakingOverResource
Detection Guideline	IF {(R1.Action violates Resource.Availability) AND (R1.Action LEADS_TO_TAKING_OVER Resource.Availability)} THEN {R1 is interacting with Resource under t23 interaction type }
Example	• R1:"During vacation, the vacation control system shall imitate the sound of occupants between times A to B using (X=TV) device" • Resource.Availability "The A/V devices are available during daytime for personal use" • Interaction: R1 will cause the unavailability of TV between A-B and this violates the resource availability constraint on having the TV available during daytime for personal use such as recording a show while away. In this case the TV is unavailable as it cannot do the two things together.
Parameters Effect	• If X was set to be an integrated/embedded audio circuit in the system then there is no interactions

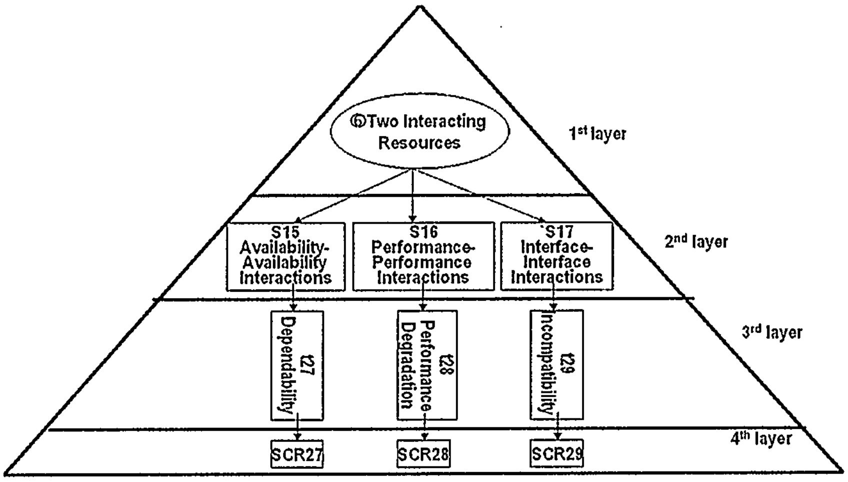
Scenario ID	SCR24
Type of Interaction	DynamicBehaviourRequirementInteractingWithResource → Action-PerformanceInteractions → PerformanceDegradation
Detection Guideline	IF {(R1.Action violates Resource.Interface) AND (R1.Action LEADS_TO_DEGRADATION Resource.Performance)} THEN {R1 is interacting with Resource under t24 interaction type }
Example	• R1 "The user can set the CD player to play stream audio tracks from the internet between times A to B using (X=dialup) connection" • Resource.Performance "Audio/Video devices have performs using high definition quality standards" • Interaction: The dialup connection has many drops in its performance. Hence, in this case the action of R1 shall affect the performance of the CD player and violates the resource performance constraints
Parameters Effect	If X is set to high speed connection such as T1 connection then there is no interaction

[4] The examples of interaction scenarios in this category are taken from smart home system which is a representative of the networked devices domain

Scenario ID	SCR25
Type of Interaction	DynamicBehaviourRequirementInteractingWithResource → Action-InterfaceInteractions → UnexpectedInputKeysBehabvior
Detection Guideline	IF {(R1.Action violates Resource.Interface) AND (R1.Action LEADS_TO_ UNEXPECTED_INPUT_ BEHAVIOUR Resource.Interface)} THEN {R1 is interacting with Resource under t25 interaction type }
Example	• R1 "To dial a voice activated number, user must pick the handset, press key (X= number key, which is an unusual input key in this case) and say the voice sample of the desired number" • Resource.Interface "Standard input interface is provided for the smart home phone interface" • Interaction: Assume a user picks the handset and press this key number, the telephone will not know if this number is part of a dialled number or it should activate the voice dialling system and hence this input might result in an unexpected behaviour"
Parameters Effect	Assign X to a special key other than number keys

Scenario ID	SCR26
Type of Interaction	DynamicBehaviourRequirementInteractingWithResource → Action-InterfaceInteractions → UnexpectedOutputDisplayBehabvior
Detection Guideline	IF {(R1.Action violates Resource.Interface) AND (R1.Action LEADS_TO_ UNEXPECTED_OUTPUT_ DISPLAY Resource.Interface)} THEN {R1 is interacting with Resource under t26 interaction type }
Example	• R1 "When the user is talking on the phone, Alert him 5 seconds before the end of every minute using (X= displaying a warning on the screen of the telephone set) technique" • Resource.Interface "Standard output interface is provided for the smart home phone interface" • Interaction: Consider a user who is storing a phone number while talking with someone on the phone. When the minute is about to finish (55 seconds), the system alerts the user and causes him to lose all his data because of the unexpected display behaviour which switches the normal screen to display the call time.
Parameters Effect	If X is changed to be an audio alarm then there is no interactions

B.5. Two Interacting Resources[5]



©Two Interacting Resources
1st layer
S15 Availability-Availability Interactions
S16 Performance-Performance Interactions
S17 Interface-Interface Interactions
2nd layer
t27 Dependability
t28 Performance-Degradation
t29 Incompatibility
3rd layer
SCR27
SCR28
SCR29
4th layer

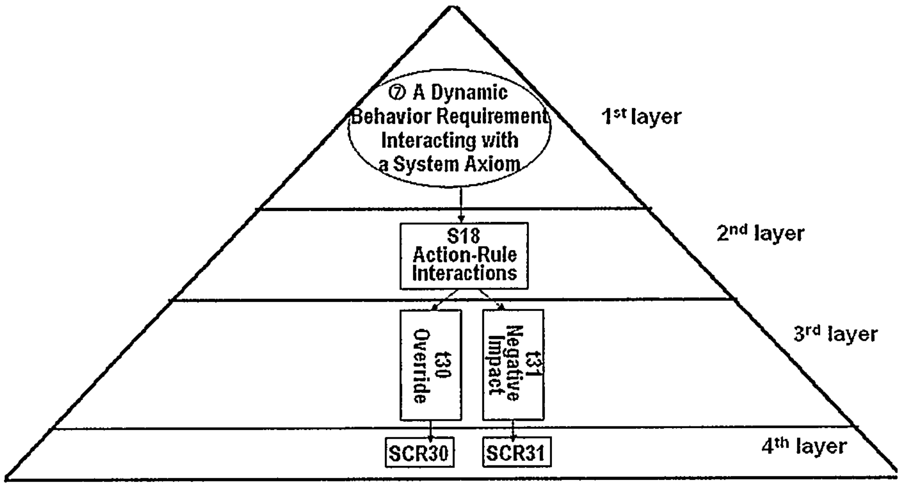
Scenario ID	SCR27
Type of Interaction	TwoInteractingResources → Availability-AvailabilityInteractions → Dependability
Detection Guideline	IF {(Resource1.Availability DEPENDS_ON Resource2.Availability)} THEN {Resource1 is interacting with Resource2 under t27 interaction type }
Example	• Resource1:Availability "The (X=natural gas) boiler shall be available more then 99.9% every year" • R2:"The natural gas regulator shall be available 100% every year" • Interaction: if the natural gas regulator fails, i.e., becomes unavailable, for any reason and the natural gas is being blocked then the boiler is not working and hence becomes also unavailable
Parameters Effect	If X is changed to be Natural gas / electric boiler then this reduces the degree of dependability between the boiler and the natural gas regulator

Scenario ID	SCR28
Type of Interaction	TwoInteractingResources→ Performance-PerformanceInteractions → PerformanceDegradation
Detection Guideline	IF {(Resource1.Performance LEADS_TO_DEGRADATION Resource2.Performance)} THEN {Resource1 is interacting with Resource2 under t28 interaction type}
Example	• Resource1:Performance "The (X=T1) Network Card, used to connect to the internet, provides best performance for connection speed" • Resource2:Performance "Audio/ devices performs using high definition quality standards" • Interaction: The performance of a CD player, which plays stream audio from the internet, is related to the performance of the T1 card. If the T1 card performance is degraded for any reason (e.g. loose connection, paths congestion) then the CD performance is also degraded.
Parameters Effect	If X is changed to be two network cards (i.e., connecting to the internet through two independent ways) then if the performance of one card is degraded then the other can compensate for that and the CD player won't feel the difference

Scenario ID	SCR29
Type of Interaction	TwoInteractingResources → Interface-InterfaceInteractions → Incompatibility
Detection Guideline	IF {(Resource1.Interface INCOMPATIBLE_WITH Resource2.Interface)} THEN {Resource1 is interacting with Resource2 under t29 interaction type }
Example	• Resource1.Interface"The TV has an X10 (which is a smart home communication protocol) compatible interface" • R2:"The VCR has (X=KONNEX, which is a smart home communication protocol) compatible interface" • Interaction: Obviously the two resources have incompatible interfaces and they cannot communicate directly with each other
Parameters Effect	If X is changed to X10 then the interaction is eliminated

[5] The examples of interaction scenarios in this category are taken from smart home system which is a representative of the networked devices domain

B.6. A Dynamic Behaviour Requirement Interacting with a System Axiom [6]

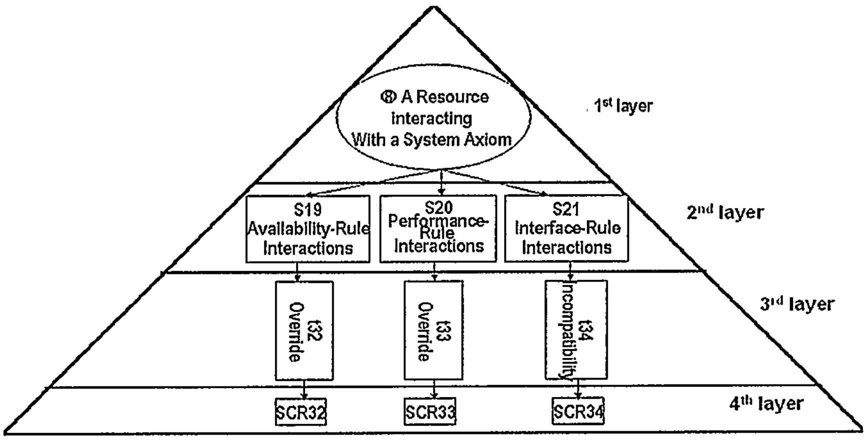

⑦ A Dynamic Behavior Requirement Interacting with a System Axiom
1st layer
S18 Action-Rule Interactions
2nd layer
t30 Override
t31 Negative Impact
3rd layer
SCR30
SCR31
4th layer

Scenario ID	SCR30
Type of Interaction	DynamicBehaviourRequirementInteractingWithSystemAxiom → Action-RuleInteractions → Override
Detection Guideline	IF {(R1.Action OVERRIDES R2.Rule)} THEN {R1 is interacting with R2 under the t30 interaction type}
Example	• R1 " for (x=unlimited times), closing of the lift door can be prevented when the user presses an open-door button" • R2 "The unserved calls are always served" • Interaction: What happens when the user keeps pressing the open-door button? In this case the action of R1 will override the rule of R2 and prevent the lift from serving unserved calls. A solution might be to force doors to close even if user is still pressing the open-doors button.
Parameters Effect	If X is changed to be a specific number then eventually, the lift doors are closed and the lift will be able to serve unserved calls.

Scenario ID	SCR31
Type of Interaction	DynamicBehaviourRequirementInteractingWithSystemAxiom → Action-RuleInteractions → NegativeImpact
Detection Guideline	IF {(R1.Action NEGATIVELY_IMPACTS R2.Rule)} THEN {R1 is interacting with R2 under the t31 interaction type}
Example	• R1 "When the lift is overloaded, then (X=the doors shall not close)" • R2 "The lift system is equipped with portable split air conditioning unit to provide air conditioned environment" • Interaction: What happens when the lift is overloaded? The action of R1 has a negative effect on rule of R2 as the open doors negatively affect the air conditioning of lift.
Parameters Effect	If X is changed to "Display an overload message and the lift shall not move" then this means that the doors are going to close but the lift won't move which preserves the air conditioning of the lift while fulfilling the safety property of not operating with an overload weight.

[6] The examples of interaction scenarios in this category are taken from the web e-commerce system which is a representative of the web domain

B.7. A Resource Interacting with A System Axiom[7]


⑧ A Resource Interacting With a System Axiom
1st layer
S19 Availability-Rule Interactions
S20 Performance-Rule Interactions
S21 Interface-Rule Interactions
2nd layer
t32 Override
t33 Override
t34 Incompatibility
3rd layer
SCR32
SCR33
SCR34
4th layer

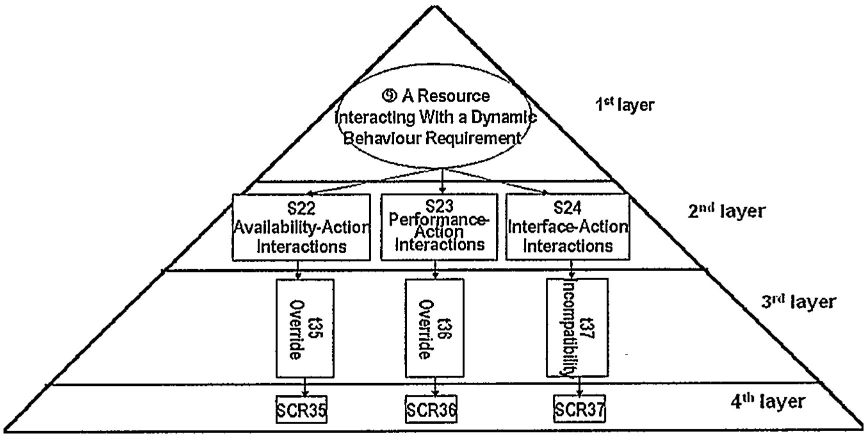
Scenario ID	SCR32
Type of Interaction	ResourceInteractingWithSystemAxiom → Availability-RuleInteractions → Override
Detection Guideline	IF {(Resource1.Availability OVERRIDES R1.Rule)} THEN {Resource1 is interacting with R1 under t32 interaction type }
Example	• Resource1.Availability "The database server is not available during (X=weekends) for maintenance purposes" • R1 "Users can access their accounts (Y=at any time)" • Interaction: The unavailability of the resource during the weekends will override the rule of R1.
Parameters Effect	If Y is changed to be weekdays and week nights then the interaction is eliminated

Scenario ID	SCR33
Type of Interaction	ResourceInteractingWithSystemAxiom → Performance-RuleInteractions → Override
Detection Guideline	IF {(Resource1.Performance OVERRIDES R1.Rule)} THEN {Resource1 is interacting with R1 under t33 interaction type }
Example	• Resource1.Performance "The response time of the database server can take up to (X=10 seconds)" • R1 "Any transaction on the website must not exceed 8 seconds" • Interaction: The performance of the database server which might take up to 10 seconds overrides the rule R1.
Parameters Effect	If X is changed to be less than 8 seconds then the interaction is resolved

Scenario ID	SCR34
Type of Interaction	ResourceInteractingWithSystemAxiom → Interface-RuleInteractions → Incompatibility
Detection Guideline	IF {(Resource1.Interface INCOMPATIBLE_WITH R1.Rule)} THEN {Resource1 is interacting with Resource2 under t29 interaction type }
Example	• Resource1.Interface"The interface of the website shall not include any online transactions pages" • R2 "The website shall be designed for (X=online shopping retailers"" • Interaction: Obviously the interface is incompatible with the website of an online shopping retailer as such an online retailer will need financial transactions webpage for customers to pay for their buys.
Parameters Effect	If X is changed to be a web site for online displaying data then there is no interaction

[7] The examples of interaction scenarios in this category are taken from the web e-commerce system which is a representative of the web domain

B.8. A Resource Interacting with a Dynamic Behaviour Requirement[8]


⑤ A Resource Interacting With a Dynamic Behaviour Requirement
1st layer
S22 Availability-Action Interactions
S23 Performance-Action Interactions
S24 Interface-Action Interactions
2nd layer
t35 Override
t36 Override
t37 Incompatibility
3rd layer
SCR35
SCR36
SCR37
4th layer

Scenario ID	SCR35
Type of Interaction	ResourceInteractingWithDynamicBehaviourRequirement → Availability-ActionInteractions → Override
Detection Guideline	IF {(Resource1.Availability OVERRIDES R1.Rule)} THEN {Resource1 is interacting with R1 under t32 interaction type }
Example	• Resource1.Availability "The database server is not available during (X=weekends) for maintenance purposes" • R1 "At time (y= 6 am every Saturday, update the contents of the website" • Interaction: The unavailability of the resource during the weekends will override the action of R1 because there are data needed from the database server in order to correctly update the website
Parameters Effect	If Y is changed to be any time during weekdays and week nights then the interaction is eliminated

Scenario ID	SCR36
Type of Interaction	ResourceInteractingWithDynamicBehaviourRequirement → Performance-ActionInteractions → Override
Detection Guideline	IF {(Resource1.Performance OVERRIDES R1.Rule)} THEN {Resource1 is interacting with R1 under t33 interaction type }
Example	• Resource1.Performance "The response time of the database server can take up to (X=10 seconds)" • R1 "after (Y=5 seconds), display the results of the operation " • Interaction: The performance of the database server which might take up to 10 seconds overrides the action of R1 and will not allow it to display correct results.
Parameters Effect	If Y is changed to be 10 seconds or more then the interaction is resolved

Scenario ID	SCR37
Type of Interaction	ResourceInteractingWithDynamicBehaviourRequirement → Interface-ActionInteractions → Incompatibility
Detection Guideline	IF {(Resource1.Interface INCOMPATIBLE_WITH R1.Rule)} THEN {Resource1 is interacting with Resource2 under t29 interaction type }
Example	• Resource1.Interface"The interface of the website shall include only (X=non interactive contents)" • R2 "When the user enters a correct user name and password, then he is logged in and a welcome message is displayed" • Interaction: Obviously the interface is incompatible with the action as the interface will not support active contents.
Parameters Effect	If X is changed to be interactive and non interactive then the interaction is eliminated

The examples of interaction scenarios in this category are taken from the web e-commerce system which is a representative of the web domain

APPENDIX C: FULL RESULTS ON THE COMPARISON IN CHAPTER 3 ON COMPARING THE PROPOSED REQUIREMENTS INTERACTION TAXONOMY WITH OTHER TAXONOMIES

This appendix contains specific and detailed results on the results of comparing the proposed interaction taxonomy presented in Chapter 3 with already existing taxonomies in the literature.

The SCR presented between brackets following individual numbers represents an interaction scenario in the proposed interaction taxonomy. For example in Table C.1, SCR8 next to 1 in the SUSC column indicates that the example number 1 is addressed by the interaction scenario SCR8 in the proposed interaction taxonomy. Also numbers in columns represent the example number in the corresponding taxonomy. For example, in Table C.1, 3 refers example 3 presented in the corresponding taxonomy which is Cameron *et al.* taxonomy.

Table C.1: Comparing proposed taxonomy to Cameron *et al.* taxonomy

		Nature of interaction (first approach: 5 categories, 22 examples)				
Cause of interaction (Second approach: 12 categories, 22 examples)		SUSC	SUMC	MUSC	MUMC	CUSY
Violations of assumptions	Naming		8 (SCR8)	10 (SCR23 or SCR26)	11 (SCR15) 12 (SCR15)	
	Data availability				16 (SCR10)	
	Administrative domain					19 (missed) 20 (missed)
	Call control	1 (SCR8) 3 (SCR14) 4 (SCR15)			14 (SCR8) 15 (SCR8)	
	Signalling protocol				13 (SCR15)	
Limitations on network support	CPE signalling	2 (SCR8 or SCR23)	7 (SCR25)			
	Funct. of Communications	5 (SCR23)				21 (missed)
Problems in distributed systems	Resource contention	2 (SCR8 or SCR23)				
	Instantiation	4 (SCR15)		9 (SCR15)	17 (SCR16)	
	Timing and race	2 (SCR8 or SCR23)	7 (SCR25)		18 (SCR16)	
	Feature support		6 (SCR15) 8 (SCR8)			
	Non-atomic operations					22 (missed)

- SUSC= Single User Single Component SUMC= Single User Multiple Component MUSC= Multiple User Single Component MUMC=Multiple User Multiple Component CUSY=Customer System
- Each cell will correspond to a category in first approach through its column and a category in second approach through its row, i.e., the cell that has number 1 in it, means that example number 1 was used to illustrate category SUSC of first approach and category call control of the second approach

Table C.2: Comparing proposed taxonomy to Kolberg *et al.* taxonomy

Interaction Category	Examples Used
Multiple Action Interaction (MAI)	1 (SCR12)
Shared Trigger Interaction (STI)	2 (SCR23)
Sequential Action Interaction (SAI)	3 (SCR11)
Missed Trigger interactions (MTI)	4 (SCR13) 5 (SCR16)

Table C.3: Comparing proposed taxonomy to Reiff-Marganiec *et al.* taxonomy

Interaction Category	Examples Used
Conditional Goals	5.1 (SCR1)
Conditional Event-Condition-Action (ECA) – Shared Trigger	5.2 (SCR10)
Conditional Event-Condition-Action (ECA) – Sequential Trigger	5.3 (SCR12)
Single Entity (SE)	SE example (SCR12)
Multiple Entity Single Branch (MESB)	MESB example (SCR1)
Single Entity Multiple Role (SEMR)	SEMR example (SCR1)
Multiple Entity Single Role (MESR)	MESR example (SCR1)
Multiple Entity Multiple Role (MEMR)	MEMR example (SCR1)
Refinement	5.4 (SCR1) or (SCR2)
Preference	Preference example (SCR11)
• Note that some categories did not include any examples for illustration such as Multiple Entities-Same Domain-Different Branches (MEDB) • The examples presented in some categories, such as the SE category, did not have a specific example numbering, so we refer to it as only example because the example is included in the body text of the category	

APPENDIX D: FULL RESULTS ON THE DEVELOPED PLUG-INS FOR IRIS IN CHAPTER 5

This appendix contains details on the developed plug-ins for IRIS from Chapter 5. It is worth mentioning that this appendix will not describe the plug-in interaction scenario because all interaction scenarios are described in details ion Chapter 3 and Appendix B. Hence, this appendix contains details for 8 plug-ins listed in Tables D.1-D.8 respectively.

Table D.1: The plug-in Functionalities Identification

Type:	STEP		
Body:	**What**	**Name**	Functionalities Identification
		Description	This Plug-in is used when a single requirement is complex and describes different functionalities. The goal of this plug-in is to simplify the parent requirement and to separate the different encapsulated functionalities into atomic functionalities that can be easily handled
		Construction	The execution of this plug-in requires the following activities: 1. For each requirement, identify complex requirements that performs more than one functionality 2. Break down the complex textual description of the requirement into atomic functionalities such that each atomic functionality can perform only one functionality 3. Go back to activity 1 until all requirements have been addressed
	When	**Problems this plug-in overcomes**	1. Solving the problem of complex requirements 2. Unclear representation of different functionalities encapsulated in one requirement 3. Lack of understanding of requirements due ambiguous complex requirements
		Expected enhancements	1. Reduced requirements ambiguity 2. Improved interaction detection between different functionalities within one requirement
	How	**Instructions**	1. This plug-in is applied prior to IRIS step 1
		Sample of application	This plug-in has been applied in a case study to identify interactions between the requirements of smart homes. Refer to Chapter 8 for an example application
Location	Since this is a STEP plug-in that is needed to be performed prior to the application of IRIS basic core steps, then this step is hooked to the hook H1		

Table D.2: The plug-in Parameters Assignment

Type:	STEP		
Body:	**What**	**Name**	Parameters Assignment
		Description	This plug-in 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).
		Construction	The execution of this plug-in requires the following activities: 1. For all system axioms and dynamic behaviour requirements, select a requirement for consideration, list it separately, and read it carefully. 2. For the requirement under consideration, identify if it has a parameterized part in its body. 3. Identify the parameterized part that needs to be replaced with a parameter 4. Replace the parameterized part of the requirement with a unique parameter (e.g., X or Y) 5. Go back to activity 3 until all requirements have been addressed.
	When	**Problems this plug-in overcomes**	1. Unclear representation of parameterized requirements 2. Unclear representation of reused requirements 3. Lack of understanding of requirements
		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 interaction detection due to interactions between the parameterized parts of the requirements
	How	**Instructions**	1. This plug-in is applied prior to step 1 of IRIS basic core 2. This plug-in is applied after the plug-in Functionalities Identification (if used) 3. This plug-in is applied to all requirements
		Sample of application	This plug-in has been applied in a case study to identify interactions between the requirements of smart homes. Refer to Chapter 8 for an example application
Location	Since this is a STEP plug-in that is needed to be performed prior to the application of IRIS basic core steps, then this step is hooked to the hook H1		

Table D.3: The plug-in Parameters

Type:	ATTR		
Body:	**What**	**Name**	Parameters
		Description	This plug-in corresponds to adding the attribute "Parameters" to the set of attributes used for representing system axioms or dynamic behaviour requirements
		Construction	The execution of this plug-in requires the following activities: 6. Add a new attribute called Parameters to the set of attributes used for representing system axioms or dynamic behaviour requirements 7. Add a new column called Parameters in the system axioms and dynamic behaviour requirements attributes identification tables created in the second step of IRIS to correspond to the attribute Parameters that was created in activity 1 8. For each requirement in any of the tables created in the second step of IRIS, list any parameters in the main body of the requirements in the new column created in activity 2 9. The parameters listed in the new Parameters column as described in activity 3 will be in the form of the parameter and its data type 10. Go back to activity 3 until all requirements have been addressed.
	When	**Problems this plug-in overcomes**	4. Solving the problem of parameterized requirements 5. Unclear representation of parameters in system axioms and dynamic behaviour requirements attributes identification tables 6. Unclear representation of the data types that parameters can have in the requirements attributes data type 7. Lack of understanding of requirements due to using unexplained parameters in the system axioms and dynamic behaviour attributes identification tables
		Expected enhancements	5. Reduced requirements ambiguity 6. Correctly dealing with parameters and data types they can have 7. Improved interaction detection due to interactions between the parameterized parts of the requirements
	How	**Instructions**	2. This plug-in is applied during IRIS step 2 3. This plug-in must be applied in conjunction of the plug-in Parameters Assignment 4. This plug-in is applied to all parameterized requirements
		Sample of application	This plug-in has been applied in a case study to identify interactions between the requirements of smart homes. Refer to Chapter 8 for an example application
Location	Since this is an ATTR plug-in that is needed to add the attribute Parameters to either system axioms or dynamic behaviour, then this plug-in is hooked to the hooks H2 or H4		

Table D.4: The plug-in Parameters Range

Type:	ATTR		
Body:	**What**	**Name**	Parameters Range
		Description	This plug-in corresponds to adding the attribute "Parameters Range" to the set of attributes used for representing system axioms or dynamic *behaviour* requirements
		Construction	The execution of this plug-in requires the following activities: 1. Add a new attribute called Parameters Range to the set of attributes used for representing system axioms or dynamic behaviour requirements 2. Add a new column called Parameters Range in the system axioms or dynamic behaviour attributes identification tables created in the second step of IRIS to correspond to the attribute Parameters Range that was created in activity 1 3. For each requirement in any of the system axiom and dynamic behaviour tables created in the second step of IRIS, identify the range of values that each parameter, listed in the parameters column, can has 4. The new Parameters Range column will contain all parameters and the range of values they can have 5. Go back to activity 3 until all requirements have been addressed.
	When	**Problems this plug-in overcomes**	1. Solving the problem of parameterized requirements 2. Unclear representation of what range of values that parameters can have in system axioms and dynamic behaviour requirements attributes identification tables 3. Lack of understanding of requirements due to using unexplained parameters in the system axioms and dynamic behaviour attributes identification tables
		Expected enhancements	1. Reduced requirements ambiguity 2. Correctly dealing with parameters and the range of values they can have 3. Improved interaction detection due to interactions between conflicting values that the parameters can have
	How	**Instructions**	1. This plug-in is applied during IRIS step 2 2. This plug-in must be applied in conjunction of the plug-in Parameters Assignment 3. This plug-in is applied to all parameterized requirements
		Sample of application	This plug-in has been applied in a case study to identify interactions between the requirements of smart homes. Refer to Chapter 8 for an example application
Location	Since this is an ATTR plug-in that is needed to add the attribute Parameters Range to either system axioms or dynamic behaviour, then this plug-in is hooked to the hooks H2 or H4		

Table D.5: The plug-in System Axioms Strategies

Type:	STEP		
Body:	**What**	**Name**	System Axioms Strategies
		Description	This plug-in is a step plug-in to identify the system axioms design and implementation strategies. This step generates a new table called "System Axioms Strategies Identification Table" to describe the available design and implementation strategies for the system axioms
		Construction	The execution of this plug-in requires the following activities: 1. For each system axiom, read carefully and understand it 2. Based on the available knowledge from experts and knowledge bases, identify the different design and implementation strategies for the system axiom under consideration 3. Construct a table that contains the information collected in activity 2
	When	**Problems this plug-in overcomes**	1. Solving the problem of identifying interactions between system axioms due to using conflicting design and implementation strategies
		Expected enhancements	1. Improved interaction detection due to interactions between conflicting design and implementation strategies
	How	**Instructions**	1. This plug-in is applied after IRIS step 2 2. This plug-in is applied to only system axioms
		Sample of application	This plug-in has not been applied in any one of the case studies in this thesis.
Location	Since this is a STEP plug-in that is needed to perform a certain step on the system axioms, then this plug-in is hooked to the hooks H2 or H4.		

Table D.6: The plug-in Availability

Type:	ATTR		
Body:	**What**	**Name**	Availability
		Description	This plug-in corresponds to adding the attribute "Availability" to the set of attributes used for representing resources requirements. The use of this plug-in will also result in a new column in the table created for the resources requirements which will contain the values regarding the availability for each resource requirement
		Construction	The execution of this plug-in requires the following activities: 1. Add the attribute Availability to the set of attributes required to represent resources 2. Add a column in the resources attributes identification table called Availability 3. In the new availability column, list availability constraints for each resource requirements
	When	**Problems this plug-in overcomes**	1. Representing availability of resources and detecting interactions that might arise from them
		Expected enhancements	1. Correctly dealing with constraints regarding the availability of resources requirements 2. Improved interaction detection due to interactions resulting from constraints on the availability of resources
	How	**Instructions**	1. This plug-in is applied during IRIS step 2 2. This plug-in is applied to only resources requirements 3. This plug-in is applied only when there are constraints regarding the availability of resources
		Sample of application	This plug-in has not been applied in any one of the case studies in this thesis.
Location	Since this is an ATTR plug-in that is needed to add the attribute Availability to resources, then this plug-in is hooked to the hooks H6		

Table D.7: The plug-in Performance

Type:	ATTR		
Body:	**What**	**Name**	Performance
		Description	This plug-in corresponds to adding the attribute "Performance" to the set of attributes used for representing resources requirements. The use of this plug-in will also result in a new column in the table created for the resources requirements which will contain the values regarding the performance of each resource requirement
		Construction	The execution of this plug-in requires the following activities: 1. Add the attribute Performance to the set of attributes required to represent resources 2. Add a column in the resources attributes identification table called Performance 3. In the new availability column, list performance constraints for each resource requirements
	When	**Problems this plug-in overcomes**	1. Representing performance of resources and detecting interactions that might arise from them
		Expected enhancements	1. Correctly dealing with constraints regarding the resources requirements performance 2. Improved interaction detection due to interactions resulting from constraints on resources performance
	How	**Instructions**	1. This plug-in is applied during IRIS step 2 2. This plug-in is applied to only resources requirements 3. This plug-in is applied only when there are constraints regarding the performance of resources
		Sample of application	This plug-in has not been applied in any one of the case studies in this thesis.
Location	Since this is an ATTR plug-in that is needed to add the attribute Performance to resources, then this plug-in is hooked to the hooks H6		

Table D.8: The plug-in Interface

Type:	ATTR		
Body:	**What**	**Name**	Interface
		Description	This plug-in corresponds to adding the attribute "Interface" to the set of attributes used for representing resources requirements. The use of this plug-in will also result in a new column in the table created for the resources requirements which will contain information regarding the interfaces of each resource
		Construction	The execution of this plug-in requires the following activities: 1. Add the attribute Interface to the set of attributes required to represent resources 2. Add a column in the resources attributes identification table called Interface 3. In the new Interface column, list Interface constraints for each resource when applicable
	When	**Problems this plug-in overcomes**	1. Representing Interfaces of resources and detecting interactions that might arise from them
		Expected enhancements	1. Correctly dealing with constraints regarding the resources Interfaces 2. Improved interaction detection due to interactions resulting from constraints on resources Interfaces
	How	**Instructions**	1. This plug-in is applied during IRIS step 2 2. This plug-in is applied to only resources requirements 3. This plug-in is applied only when there are constraints regarding the interfaces of resources
		Sample of application	This plug-in has not been applied in any one of the case studies in this thesis.
Location	Since this is an ATTR plug-in that is needed to add the attribute Interface to resources, then this plug-in is hooked to the hooks H6		

APPENDIX E: FULL RESULTS FROM CHAPTER 8 ON THE DETECTED INTERACTIONS IN SMART HOMES

This appendix presents all the detected interactions from applying IRIS in the smart homes case study presented in Chapter 8. The aim is to provide clarification on how each interaction was detected and an example scenario of interaction for illustration purposes. Table E.1 uses the following abbreviations:

- ID: A unique interaction ID for each interaction
- Policies: Identify the two simple policies that interact
- Analysis: Lists what analysis procedure was used to detect the interaction
- Interaction: Lists an example scenario of a possible interaction, and suggests a resolution. The scenario listed might not be the only possible interaction scenario and there might be other situations in which the two policies interact. Similarly, the solution is only a suggestion and the manufacturer and occupants might prefer other solutions.

The procedure described in section 8.5.7 explains in details how interactions are detected. This procedure was also used to detect the interactions listed in this appendix. Equal priorities of all simple policies were assumed during the detection step.

Table E.1: Detected interactions using IRIS and suggested solutions

ID	Policies	Analysis	Interaction
I1	P1.1, P1.2	Linked events E1, E2	Type: The action of P1.1 overrides the action of P1.2. Scenario: A thief opens the window and once he is in, he quickly deactivates the alarm using the alarm switch thus making the alarm appear as a system glitch. Solution: Freeze the security alarm control panel as soon as alarm is triggered until a PIN is provided.
I2	P1.1, P1.3	Linked events E1, E3	Type: The action of P1.1 overrides the action of P1.3. Scenario: A thief opens the door and once he is in, he quickly deactivates the alarm using the alarm switch thus making the alarm appear as a system glitch. Solution: Freeze the security alarm control panel when alarm triggers until a PIN is provided.
I3	P1.1, P1.4	Linked events E1, E4	Type: The action of P1.1 overrides the action of P1.4. Scenario: A thief is in the house and once he sees the PIR, he quickly deactivates the alarm using the alarm switch thus making the alarm appear as a system glitch. Solution: Freeze the security alarm control panel when alarm triggers until a PIN is provided.
I4	P1.1, P1.5	Linked events E1, E5	Type: The action of P1.1 overrides the action of P1.5. Scenario: A thief is in the house, once he feels the alarm is triggered by the pressure pads, he quickly deactivates the alarm using the alarm switch thus making the alarm appear as a system glitch. Solution: Freeze the security alarm control panel when alarm triggers until a PIN is provided.
I5	P1.1, P3.2	Linked events E1, E9	Type: The action of P3.2 overrides the action P1.1. Scenario: P1.1 activates the security alarm to secure the house, while P3.2 can still override P1.1 and the door can be opened. For example, if a thief is inside the house and wants to get out, he can just press the open door switch to get out. Another example, if an occupant opens the doors using P3.2 then it will falsely trigger the alarm. Solution: If alarm is activated using P1.1, a PIN is required before executing P3.2.
I6	P1.1, P10.1	Linked events E1, E7	Type: The action of P10.1 negatively impacts the action of P1.1. Scenario: Occupant schedules windows to open in a certain place at a certain time and while this action is executing, occupant without knowing that windows are opening, activates alarm and hence falsely triggers the alarm. Solution: Ask the occupant to secure open doors and windows before executing P1.1.
I7	P1.1, P12.1	Linked events E1, E18	Type: The action of P12.1 negatively impacts the action of P1.1. Scenario: Occupant A comes home and deactivates alarm. Occupant B at work can't remember if he activated alarm or not, so he calls and activates alarm using remote access module. Occupant A opens a window and alarm triggers. Solution: Information about the last deactivation of the security alarm is provided over the phone to the person who wants to use the remote access module to set the alarm.
I8	P1.2, P3.2	Linked events E2, E9	Type: The action of P3.2 overrides the action of P1.2. Scenario: An occupant unlocks the main door using P3.2 while the alarm system is active. If this disables the alarm, to avoid a false trigger, and the main door is opened, then a thief could open a window in the basement at the same time and the alarm will not be triggered. Thus P1.2 action is overridden by P3.2 action. Solution: When P3.2 opens the door, then only the door is excluded from triggering the alarm.
I9	P1.2, P10.1	Linked events E2, E7	Type: The action P10.1 negatively impacts the action of P1.2. Scenario: If the occupant sets a time when the windows open automatically while the security alarm is active then the action of P10.1 will falsely trigger the alarm. Solution: The occupant is notified to cancel scheduled windows opening times before alarm is allowed to be active.
I10	P1.2, P12.1	Linked events E2, E18	Type: The action of P12.1 overrides the action of P2.1. Scenario: A thief breaks into the house through a window. One second later, an occupant uses the remote access module to cancel the alarm as he is on his way home. The alarm trigger looks like a system glitch and thief escapes. Solution: If the alarm is triggered then the remote access module cannot disable the alarm system.
I11	P1.3, P3.2	Linked events E3, E9	Type: The action of P3.2 negatively impacts the action of P1.3. Scenario: Occupants open the main door using the interior main door switch while the alarm is active. This falsely triggers an alarm. Solution: The system shall ask for a PIN before opening the door if alarm is active.
I12	P1.3, P3.3	Linked events E3, E10	Type: The action of P3.3 negatively impacts the action of P1.3. Scenario: Steam and smoke from cooking causes the G/H/S detector to trigger while the alarm is active. The system opens the main door according to P3.3. Thus the intruder alarm triggers falsely. Solution: Intruder alarm is disabled when the G/H/S detector triggers.
I13	P1.3, P12.1	Linked events E3,E18	Type: The action of P12.1 overrides the action of P1.3. Scenario: A thief breaks into the house through the door and the alarm is triggered. One second later, an occupant uses the remote access module to deactivate the alarm as he is on his way home. Then the thief can get away as the alarm trigger looks like a system glitch. Solution If alarm is triggered then the remote access module cannot disable the security alarm.

Table E.1- Continued: Detected interactions using IRIS and suggested solutions

I14	P1.4, P3.2	Linked events E4, E9	Type: The action of P3.2 overrides the action of P1.4. Scenario: An occupant unlocks the main door using P3.2 while the alarm system is active. To avoid false triggering o alarm, the system disables the alarm then opens main door. If during this time a thief breaks in from the upper floo and his movements are detected by a PIR, then alarm will not be triggered because it is temporarily disabled. Solution: When P3.2 opens the main door, then only the main door is excluded from triggering the alarm.
I15	P1.4, P8.2	Same trigger event E4	Type: The action of P8.2 negatively impacts the action of P1.4. Scenario: An occupant sets the same part of the house (e.g. hallway) for both lights (in P8.2) and security (P1.4) t check for a positive PIR signal. The occupant gets up at night and the PIR sensor detects his movements thus th system activates lights but also activates the alarm which in this case is triggered falsely by the occupant. Solution: Don not allow occupants to set the same area for lights increase and security check.
I16	P1.4, P12.1	Linked events E4, E18	Type: The action of P12.1 overrides the action of P1.4. Scenario: A thief breaks into the house and triggers the alarm by a positive PIR signal. But an occupant uses th remote access module to deactivate the alarm as he is on his way home. Then the thief can get away as the alar trigger looks like a system glitch. Solution If alarm is triggered then the remote access module cannot disable the security alarm.
I17	P1.5, P3.2	Linked events E5, E9	Type: The action of P3.2 overrides the action of P1.5. Scenario: An occupant unlocks the main door using P3.2 while the alarm system is active. To avoid false triggering o alarm, the system disables alarm then opens main door. If during this time a thief breaks in from upper floor and hi movements are detected by pressure pads, then alarm will not be triggered because it is temporarily disabled. Solution: When P3.2 opens the main door, then only the main door is excluded from triggering the alarm.
I18	P1.5, P8.2	Linked events E4, E5	Type: The action of P8.2 negatively impacts the action of P1.4. Scenario: An occupant sets the same part of the house (e.g. hallway) for both lights and security to check for PIR an pressure pads signals. The occupant gets up at night and activates lights by PIR but also walks on the pressure pad in the hallway thus triggers alarm by himself falsely. Solution: Do not allow occupant to set the same area for lights increase and pressure pads security check.
I19	P1.5, P12.1	Linked events E5, E18	Type: The action of P12.1 overrides the action of P1.5 Scenario: A thief breaks into the house and triggers the alarm by the pressure pads. But an occupant uses th remote access module to deactivate the alarm as he is on his way home. Then the thief can get away as the alar trigger looks like a system glitch. Solution If alarm is triggered then the remote access module cannot disable the security alarm.
I20	P2.1, P2.2	Linked events E6, E7	Type: The action of P2.1 overrides the action of P2.2. Scenario: If an occupant presses the deactivate switch while P2.2 is executing, then P2.1 will cancel the action o P2.2 before completion. Note that both functionalities are of equal priorities. This might be important if a child is th one who deactivated the vacation control. Solution: P2.1 is of higher priority and a PIN is required before the actual deactivation of the vacation control.
I21	P2.1, P2.3	Linked events E6, E7	Type: The action of P2.1 overrides the action of P2.3. Scenario: If an occupant presses the deactivate switch while P2.3 is executing, then P2.1 will cancel the action o P2.3 before completion. Note that both functionalities are of equal priorities. This might be important if a child is th one who deactivated the vacation control. Solution: P2.1 is of higher priority and a PIN is required before the actual deactivation of the vacation control.
I22	P2.1, P10.1	Linked events E6, E7	Type: P 10.1 negatively impacts the action of P2.1. Scenario: P10.1 is a security hole that negatively impacts the intended purpose of P2.1 which is to keep the hous safe during extended periods of absence. For example, an occupant activates vacation control (P2.1) as a protectio of the house while away but P10.1 can still open windows and thus negatively impact the intended purpose of P2.1 Solution: Ask the occupant to cancel scheduled window opening before activating vacation control.
I23	P2.2, P4.1	P2.2 interacts with system axiom P4.1	Type: The rule of P4.1 overrides the action of P2.2 Scenario: The vacation control is on and P2.2 is executing. The occupant comes home and forgets to turn th vacation control off. The occupant then tries to use the remote control to turn the TV off. If the TV is turned off, ther the rule P4.1 overrode the action of P2.2. If not then P4.1 is violated by P2.2. Solution: The occupant must turn the vacation control off first before being able to use the remote control.
I24	P2.2, P4.2	Same trigger event E7	Type: The action of P4.2 overrides the action of P2.2. Scenario: An occupant sets the action of P2.2 to turn on the TV at time X for 60 minutes while the action of P4.2 wa set to turn off the TV at the same time X. A similar scenario can also occur if the defined times overlap. Solution: Manual settings by occupants for TV, such as P4.2, are cancelled when the vacation control is active.
I25	P2.2, P5.2	P2.2 interacts with system axiom P5.2	Type: The action of P2.2 overrides the rule of P5.2. Scenario: An occupant sets the TV to Y volume. The next day he lowers the max audio level in P5.2 below Y. Ther he activates the vacation control (thus activating P2.2) and leaves. When P2.2 starts the TV with the last setting o volume which is Y, it violates P5.2. Solution: The vacation control starts the TV with a volume below the allowed max audio level.

Table E.1- Continued: Detected interactions using IRIS and suggested solutions

I26	P2.2, P9.1	Same trigger event E7	Type: The action of P9.1 negatively impacts the action of P2.2. Scenario: P9.1 will be a security hole that negatively impacts the intended purpose of P2.2. For example, if the predefined areas of curtain and blinds are the same as the location of TV and the predefined time is the same for P2.2 and P9.1, then this will enable by-passers see that no one is watching TV. Solution: Disable the opening of curtains and blinds when the vacation control is opening the TV. However when the TV is not on then curtains and blinds can be opened/closed to give impression that occupants are home.
I27	P2.2, P9.2	Linked events E7, E15	Type: The action of P9.2 negatively impacts the action of P2.2. Scenario: P9.2 will be a security hole that negatively impacts the intended purpose of P2.2. For example, if the predefined area of curtains and blinds are the same as the location of TV and the predefined time is the same for P2.2 and P9.1, then this will let anyone looking from windows know that no one home watching TV Solution: Disable the opening of curtains and blinds when the vacation control is opening the TV.
I28	P2.2, P10.1	Same trigger event E7	Type: The action of P10.1 negatively impacts the action of P2.2. Scenario: The action of P10.1 will be a security hole that counteracts the intended purpose of vacation control P2.2 which is to keep the house safe during extended periods of absence. For example, an occupant activates vacation control (P2.2) and windows control (P10.1) then leaves. Solution: Disable the opening of windows when the vacation control is activated.
I39	P2.2, P12.1	Linked events E7, E18	Type: The action of P12.1 overrides the action of P2.2 Scenario: Occupant A activates vacation control (and thus P2.2) and leaves. Occupant B calls and use remote access module (P12.1) to turn off the TV or even cutoff the power to it. Hence overriding the action of P2.2 Solution: Remote access module cannot control TV when vacation control is active
I30	P2.3, P6.1	Linked events E7, E12	Type: The action of P2.3 negatively impacts the action of P6.1. Scenario: An occupant chooses low Temperature for P6.1 and also chooses to turn on all lights with high watts. The heat radiated from the light bulbs swill affect the decrease of temperature and requires more power for the HVAC. Solution: With low temperature settings, use medium light intensity or low power lamps.
I31	P2.3, P6.2	Same trigger event E7	Type: The action of P2.3 negatively impacts the action of P6.2. Scenario: An occupant chooses low Temperature for P6.2 and also chooses to turn on all lights with high watts. The heat radiated will affect the decrease of temperature and requires more power for the HVAC. Solution: With low temperature settings, use medium light intensity or low power lamps.
I32	P2.3, P8.1	Linked events E7,E13	Type: The action of P8.1 overrides the action of P2.3. Scenario: Vacation control is on and occupant comes home. P2.3 has finished (after 60 min) and starts shutting off lights while occupant uses light dimmer to increase the light intensity and the lights might not respond to this request. Solution: Occupant must turn the vacation control off once s/he enters the house before getting control over lights.
I33	P2.3, P8.2	Linked events E4, E7	Type: The action of P2.3 overrides the action of P8.2. Scenario: The occupant comes home and forgets to deactivate vacation control (P2.3). At night P2.3 triggers and switches on the lights for 59 min. Then occupant gets up to go to bathroom hence triggering P8.2 which will increase light to the max over 2 minutes. But P2.3, after first minute, shuts down lights because the 60 minutes have elapsed. Solution: Occupant must turn off the vacation control once s/he gets home before other policies are active again.
I34	P2.3, P8.3	Linked events E7, E14	Type: The action of P8.3 overrides the action of P2.3. Scenario: An occupant has both P2.3 and P8.3 active. P2.3 triggers and switches the lights on. After 15 minutes P8.2 switches off lights because no one is home. Thus P8.3 switches off lights after 15 min. (not 60 min as in P2.3). Solution: Disable other light control policies when vacation control is activated.
I35	P2.3, P8.4	Linked events E7, E15	Type: The action of P8.4 overrides the action of P2.3. Scenario: P2.3 is triggered and switches on the lights for 60 min. At the end of the 60 minutes, the system starts shutting off the lights. At the same instant, the night begins and P8.4 starts to open lights while they are being shut off. Therefore the lights are not shutoff and P3.2 action is not completed Solution: Disable other light control policies when vacation control is activated.
I36	P2.3, P9.1	Same trigger event E7	Type: The action of P9.1 negatively impacts the action of P2.3. Scenario: P9.1 will be a security hole that negatively impacts the intended purpose of P2.3. For example, if the predefined area is the same for curtains and lights and the predefined time is also the same for P2.3 and P9.1, then anyone looking from windows know that no one home opening/closing the lights. Solution: Close curtains/blinds during times when vacation control is active and lights are about to be turned on/off.
I37	P2.3, P9.2	Linked events E7, E15	Type: The action of P9.2 negatively impacts the action of P2.3. Scenario: P9.2 will be a security hole that negatively impacts the intended purpose of P2.3. For example, If the predefined area is the same for curtains and lights and the predefined time is also the same for P2.3 and P9.2. Then anyone looking from windows know that no one home opening/closing the lights. Solution: Close curtains and blinds during times when vacation control is active and the lights are about to be turned on/off. However, curtains and blinds can be opened during other times to give impression that occupants are home.
I38	P2.1, P12.1	Linked events E6, E18	Type: The action of P12.1 overrides the action of P2.1 Scenario: Occupant A deactivates vacation control when she comes home. Occupant B is away and cannot remember if he activated the vacation control or not, so he calls and uses remote access module (P12.1) to turn on vacation control. Occupant A at home looses control over lights and TV. Solution: State last activation/deactivation information of security alarm over the phone to the person who wants to use the remote access module to set the alarm.

Table E.1- Continued: Detected interactions using IRIS and suggested solutions

I39	P2.3, P10.1	Same trigger event E7	Type: The action of P10.1 negatively impacts the action of P2.3. Scenario: P 10.1 will be a security hole that counteracts the intended purpose of vacation control P2.3 which is tc keep the house safe while extended periods of absence. For example, an occupant activates vacation control (P2.3) and windows control (P10.1) then leaves. Solution: Disable the opening of windows when the vacation control is activated.
I40	P2.3, P12.1	Linked events E7, E18	Type: The action of P12.1 overrides the action of P2.3. Scenario: Occupant A activates vacation control (and thus P2.3) and leaves. Occupant B calls and uses the remote access module (P12.1) to turn off all lights in the home, hence overriding the action of P2.3. Solution: Remote access module cannot control lights when vacation control is active.
I41	P3.1, P3.3	Linked events E8, E10	Type: The action of P3.3 overrides the action of P3.1. Scenario: The only occupant at home shuts the main door expecting it to lock automatically according to P3.1 and leaves. One minute later, G/H/S triggers and opens door according to P3.3, thus leaving house vulnerable to anyone. Solution: Only open the main door when there is someone inside (movements can be detected using PIR).
I42	P3.1, P12.1	Linked events E8, E18	Type: The action of P12.1 is used to override the action of P3.1. Scenario: Occupant A is leaving and shuts the doors behind her expecting it to lock automatically. While shutting, occupant B calls and opens the main door lock. Thus house is vulnerable. Solution: Critical parts of the house like the main door cannot be controlled by remote access module.
I43	P3.2, P6.1	Linked events E9, E12	Type: The action of P3.2 negatively impacts the action of P6.1. Scenario: When occupants use P3.2 to open the main door while P6.1 is triggered trying to raise the temperature o the house. The open door will affect the increase of temperature if left open for a long time. Solution: Close the door after some time units to maintain the temperature of the home.
I44	P3.2, P6.2	Linked events E7, E9	Type: The action of P3.2 negatively impacts the action of P6.2. Scenario: When occupants use P3.2 to open the main door while P6.2 is triggered trying to raise the temperature o the house. The open door will affect the increase of temperature if left open for a long time. Solution: Close the door after some time units to maintain the temperature of the home.
I45	P3.2, P12.1	Linked events E9, E18	Type: The action of P12.1 overrides the action of P3.2 Scenario: Occupant A presses unlock door interior switch to unlock door. Occupant B calls and uses remote access module to lock door, as he suspects it was left open. Occupant A tries to open the door but it does not respond. Solution: The interior switch has higher priority and still opens the house's main door lock.
I46	P3.3, P6.1	Linked events E10, E12	Type: The action P3.3 negatively impacts the action of P6.1. Scenario: P6.1 tries to raise the house's temperature. G/H/S is triggered by mistake (e.g. battery fault or short circuit) and opens the door according to P3.3. If left for a long time, it will affect the ability of P6.1 to increase temperature. Solution: Close door after some time units, but only if the G/H/S alarm has stopped.
I47	P3.3, P6.2	Linked events E7, E12	Type: The action P3.3 negatively impacts the action of P6.2. Scenario: P6.2 tries to raise the house's temperature. G/H/S is triggered by mistake (e.g. battery fault or short circuit) and opens the door according to P3.3. If left for long time, it will affect the ability of P6.2 to increase temperature. Solution: Close door after some time units, but only if the G/H/S alarm has stopped.
I48	P3.3, P12.1	Linked events E10, E18	Type: The action of P12.1 overrides the action of P3.3. Scenario: G/H/S triggers because of a fire and opens the main door and hence occupant A tries to get out. Occupant B calls and uses the remote access module to close the main door (as he suspects it might have accidentally been left open). Occupant A is then stuck inside. Solution: Remote access module is disabled in case of emergencies, such as fire.
I49	P4.1, P4.2	P4.2 interacts with system axiom P4.1	Type: The action of P4.2 overrides the rule of P4.1 Scenario: Occupant A uses the remote control to switch on the TV while at the same time P4.2 is scheduled by occupant B to turn off the TV. The TV shuts according to the action of P4.2 and hence P4.1 rule has been violated. Solution: Assign higher priority for the system axiom P4.1.
I50	P4.1, P5.1	P5.1 interacts with system axiom P4.1	Type: P4.1 rule overrides the action of P5.1 Scenario: Some A/V devices (like TVs) take several seconds before they are actually on. If during this time, ar occupant uses the remote control to raise volume very high (using P4.1), then the device will have different volume setting when it is actually on than the preset sound level defined in P5.1. For example, a parent presets TV volume tc start at volume X but a child increases volume (using P4.1) to the maximum before the actual start of the TV device. Solution: Assign higher priority for the action of P5.1.
I51	P4.1, P5.2	Two Interacting system axiom P4.1, P5.2	Type: The rule of P4.1 overrides the rule of P5.2. Scenario: An occupant tries to use the remote control of an A/V device to go beyond the maximum preset audio leve of the house. Note that this might be a multi-user environment like parents and children. Solution: Assign higher priority for the rule of P5.2.

Table E.1- Continued: Detected interactions using IRIS and suggested solutions

I52	P4.1, P12.1	P12.1 interacts with System axiom P4.1	Type: The action of P12.1 overrides the rule of P4.1. Scenario: Occupant A uses the remote control to turn on the TV (using P4.1 rule). Occupant B calls and uses the remote access module (P12.1) to turn off all A/V devices (as he suspects he forgot to switch them off when he left). Solution: State last activation information over phone before turning off any A/V device.
I53	P4.2, P5.2	P4.2 interacts with system axiom P5.2	Type: The action of P4.2 overrides the rule of P5.2. Scenario: Occupant A (e.g. parent) uses P5.2 to set a relatively low maximum audio level for the house. Every family member uses P4.2 to turn on an A/V device at overlapping time settings. The combined volume of the several audio devices will exceed the max volume allowed for the home. Solution: Assign higher priority of P5.2 and do not allow combined volumes of A/V devices to exceed the max. limit.
I54	P4.2, P12.1	Linked events E7, E18	Type: The action of P12.1 overrides the action of P4.2. Scenario: Occupant A sets the VCR to turn on and record a show. Occupant B calls from work to completely shut down all A/V devices as he suspects that he left one of them on. This prevents P4.2 action from ever executing. Solution: State to user over phone if there are any A/V devices affected by power cut off or scheduled to work later.
I55	P5.1, P5.2	P5.1 interacts With system Axiom P5.2	Type: The action of P5.1 overrides the rule of P5.2. Scenario: Occupant A (e.g. parent) sets the maximum audio level of the house to X. Occupant B (e.g. child) presets audio of TV and CD to levels that when combined (i.e. added to each other) will exceed max audio level of house X. Solution: Assign higher priority to P5.2 and do not allow combined volumes of A/V devices to exceed the max. limit.
I56	P5.1, P12.1	Linked events E11, E18	Type: The action of P12.1 overrides the action of P5.1. Scenario: Occupant A (e.g. parent) receives a phone call from a neighbor that TV is too loud. The parent calls and uses remote access module to lower volume of all A/V devices. At the same time, occupant B (e.g. child) turns on a CD recorder expecting a certain audio level, as specified in P5.1, but the parent has lowered volume using P12.1,. Solution: State over the phone if there are A/V devices affected by lowering the volume. In all cases, a priority assignment is needed in case parent still proceed with lowering the volume.
I57	P5.2, P12.1	P12.1 interacts with system axiom P5.2	Type: The action of P12.1 overrides the rule of P5.2. Scenario: Occupant A uses P5.2 to set a low maximum audio level Y for the house. Occupant B calls and uses the remote access module to activate several A/V devices that have audio levels which when combined (i.e. added to each other) will be greater than Y. Solution: Assign higher priority for P5.2 and do not allow combined volumes of A/V devices to exceed the max limit.
I58	P5.2, P16.1	P16.1 interacts With system axiom P5.2	Type: The action of P16.1 overrides the rule of P5.2. Scenario: The already existing audio level of the house is almost at maximum. Occupant A activates various loud appliances like the food processor and blender. Although appliances are not A/V devices they increase the noise level of the house and violates the intended purpose of P5.2 which is to keep the house below a certain noise level. Solution: Reduce the volume of an A/V device to compensate for the additional noise of appliances.
I59	P6.1, P6.2	Linked events E7, E12	Type: Action of P6.2 overrides the action of P6.1. Scenario: Occupant A uses P6.1 to preset X as the temperature of the house for the whole day. Occupant B is not aware of the preset value of X and sets another temperature Y during day using P6.2. If X is different from Y then later one might override the prior temperature. Solution: Do not allow different temperature settings in overlapping time intervals.
I60	P6.1, P10.1	Linked events E7, E12	Type: The action of P10.1 negatively impacts the action of P6.1. Scenario: P10.1 opens the windows. If the outside temperature is low then the opened windows will negatively impact P6.1 and prevent the HVAC unit from keeping the room temperature at the predefined temperature setting. Solution: The system checks for the outside temperature and if the temperature affect the room temperature if the windows are open then the occupant is prompted to choose between either one of the policies.
I61	P6.1, P12.1	Linked events E12, E18	Type: The action of P12.1 overrides the action of P6.1. Scenario: The temperature inside the house gets low and P6.1 triggers. An occupant calls and uses the remote access module to shutdown the HVAC unit before completing its work. A similar scenario can occur when the occupant calls and uses the remote access module to open the windows. Solution: User is informed over phone if temperature is affected by the remote access module action
I62	P6.2, P10.1	Same trigger Event E7	Type: The action of P10.1 negatively impacts the action of P6.2. Scenario: P10.1 opens the windows. If the outside temperature is low then the opened windows will negatively impact P6.2 and prevent the HVAC unit from getting the room temperature to the predefined temperature setting. Solution: The system checks for the outside temperature and if the temperature affect the room temperature if the windows are open then the occupant is prompted to choose between either one of the policies.
I63	P6.2, P12.1	Linked events E7, E18	Type: The action of P12.1 overrides the action of P6.2. Scenario: P6.2 triggers to increase/decrease the temperature to the predefined settings. The occupant calls and uses the remote access module to shutdown the HVAC unit before the predefined temperature has been reached. Or the occupant calls and uses the remote access module to open the windows. Solution: The occupant is informed over the phone if the temperature is affected by the action of the remote access module and is asked to confirm the action.

Table E.1- Continued: Detected interactions using IRIS and suggested solutions

I64	P8.1, P8.2	Linked events E4, E13	Type: The action of P8.1 overrides the action of P8.2. Scenario: An occupant wakes up at night and P8.2 triggers to increase the light to a maximum over 2 minutes. The occupant uses light dimmer to decrease the lights intensity (P8.1). Thus both are not able to execute at same time. Solution: Assign higher priority to manual light dimmer and terminate the action of P8.2 if light dimmer is used.
I65	P8.1, P8.4	Linked events E13, E15	Type: The action of P8.1 overrides the action of P8.4. Scenario: An occupant uses the light dimmer to decrease the light's intensity (P8.1) but at the same time the night starts and P8.4 tries to turn on and increase the intensity of lights to the specified maximum. Solution: Assign higher priority to manual light dimmer and terminate the action of P8.4 if light dimmer is used.
I66	P8.1, P12.1	Linked E13, E18	Type: The action of P12.1 overrides the action of P8.1. Scenario: Occupant A gets home and uses the light dimmer to increase the light intensity (P8.1). Occupant B calls and uses the remote access module to switch off all lights as he suspects he left them on. Solution: Assign higher priority to manual light dimmer and terminate the action of P12.1 if light dimmer is used.
I67	P8.2, P12.1	Linked events E4, E18	Type: The action of P12.1 is used to cancel the action of P8.2. Scenario: Occupant A gets up at night and thus triggering P8.2 to increase light intensity. Occupant B calls and uses the remote access module to switch off all lights as he suspects he left them on not knowing that occupant A is home Solution: Inform the user over the phone that someone is at home and ask for confirmation before executing actions.
I68	P8.3, P12.1	Linked events E14, E18	Type: The action of P8.3 overrides the action of P12.1. Scenario: An occupant uses P12.1 to switch on the lights in the garage just before his arrival. For some reason he is 15 minutes late (P8.3 has now switched off the lights) and when he gets into the garage the lights are off. Solution: The occupant is informed over the phone if he wants to double the time before P8.3 switches off the lights. Then the occupant can decide to accept or not.
I69	P8.4, P12.1	Linked events E15, E18	Type: The action of P12.1 overrides the action of P8.4. Scenario: Occupant A activates P8.4 when she gets home and takes a shower. Night begins and P8.4 switches on the lights including the bathroom light. Occupant B calls and uses the remote access module (P12.1) to switch off all lights not knowing that occupant A is home. Solution: Inform user over the phone that someone is at home and ask for confirmation before executing the actions.
I70	P9.1, P9.2	Linked events E7, E15	Type: The action of P9.1 overrides the action of P9.2. Scenario: Occupant A uses P9.1 to set the curtains/blinds to open at 6 PM (same time night starts). Occupant B sets curtains/blinds to close when night begins using P9.2. Solution: Assign higher priority to either one of them.
I71	P9.1, P12.1	Linked events E7, E18	Type: The action of P12.1 overrides the action of P9.1. Scenario: Occupant A sets the curtains to open at time X using P9.1. When time X comes, the curtains start opening. Occupant B calls and uses the remote access module to close curtains thus cancelling action of P9.1. Solution: Inform user over phone of affected policies actions (P9.1) and ask for confirmation before execution.
I72	P9.2, P12.1	Linked events E15, E18	Type: The action of P12.1 overrides the action of P9.2 Scenario: Occupant A sets the curtains to open in the early morning using P9.2 and when the day begins the curtains open. Occupant B, who spent the night at work, calls and uses the remote access module to close curtains (as he suspects he might have left them open) thus cancelling action of P9.2. Solution: Inform the user over phone of affected policies actions (P9.2) and that there is someone at home who had set a new policy that uses P9.2. Then user is asked for confirmation before proceeding to execute any commands.
I73	P10.1, P12.1	Linked events E7, E18	Type: The action of P12.1 overrides the action of P10.1. Scenario: Occupant A sets the windows to open at time X using P10.1. At time X, the windows starts opening. Occupant B calls and uses the remote access module to close all windows thus cancelling action of P10.1. Solution: Inform the user over the phone of affected policies actions (P10.1) and that there is someone at home who set a new policy that uses P10.1. Then user is asked for confirmation before proceeding to execute any commands.
I74	P11.1, P12.1	Linked events E17, E18	Type: The action of P12.1 overrides the action of P11.1. Scenario: Occupant A takes a shower and leaves without tightly closing the water tap. The water starts filling the tub till it reaches 75%. P11.1 triggers and starts closing the water tap. Occupant B calls and uses the remote access module to open the water tap in shower for 10 minutes to fill the tub before his arrival and thus flooding bathroom. Solution: Assign higher priority to P11.1.
I75	P12.1, P13.1	P12.1 interacts With system axiom P13.1	Type: The action of P12.1 overrides the rule of P13.1. Scenario: When one occupant calls and uses the remote access module (P12.1) for a long time this violates the presence of a telephone line enforced by P13.1 as they both use the same telephone line. Solution: Do not allow extended use of the remote access module beyond a certain time limit.
I76	P12.1, P13.2	Same trigger event E18	Type: Next state non-determinism between P12.1 and P13.2. Scenario: The system will have a next state non-determinism if the occupant assigns the number of rings to activate the remote access module and the answer machine to be the same. The system does not know which next state it should transfer to: the answer machine or the remote access module. Solution: Assign higher priority to either one of them.

Table E.1- Continued: Detected interactions using IRIS and suggested solutions

I77	P12.1, P14.1	Linked events E7, E18	Type: The action of P12.1 overrides the rule of P14.1. Scenario: A parent uses P14.1 to prevent any activation of the stove while s/he is out. A child uses a cell phone to call the home phone number and uses the remote access module to activate the stove. Solution: Assign higher priority to P14.1.
I78	P12.1, P14.2	Linked events E10, E18	Type: The action of P12.1 overrides the action of P14.2. Scenario: The G/H/S triggers and shuts down the stove to prevent any fire (P14.1). Occupant A .calls and uses the remote access module (P12.1) to turn on the oven to be heated until he gets home. Solution: Assign higher priority to P14.2.
I79	P12.1, P15.1	Linked events E18, E19	Type: The action of P12.1 overrides the action of P15.1. Scenario: Occupant A is cooking and the humidity sensor triggers and turns on the kitchen fan (P15.1). Occupant B calls, not knowing that occupant A is home, and uses the remote access module to shutdown all kitchen appliances including the kitchen fan as he suspects he might have accidentally left them on. Solution: Inform user over phone of affected policies actions (P15.1) and that there is someone at home.
I80	P12.1, P16.1	P12.1 interacts With system axiom P16.1	Type: The action of P12.1 overrides the action of P16.1. Scenario: Occupant A at home uses the remote control to run the food processor (P16.1). Occupant B, not knowing that occupant A is home, calls and uses the remote access module to switch off all kitchen appliances as he suspects he might has left something switched on. Solution: Inform the occupant over the phone of affected policies actions (P16.1) and that there is someone at home. Then the occupant is asked for confirmation before executing any commands.
I81	P14.1, P16.1	P14.1 interacts With system axiom P16.1	Type: The rule of P16.1 overrides the action of P14.1 Scenario: P16.1 enforces that any appliances including the stove can be controlled by the remote control. What happens if a parent uses P14.1 to switch off the stove while being away and a child finds the remote control and use it to turn on the stove? If the stove turns on then P14.1 was cancelled; if not, then P16.1 was violated. Solution: Assign higher priority to P14.1.
I82	P14.2, P16.1	P14.2 interacts With system axiom P16.1	Type: The rule of P16.1 overrides the action of P14.2. Scenario: P16.1 enforces that any appliances including the stove can be controlled by the remote control. What happens if the G/H/S triggers (maybe falsely) P14.2 to switch off the stove and a child finds the remote control and insists on using it to turn on stove. If the stove turns on then P14.2 was overridden. If not, then P16.1 was violated. Solution: Assign higher priority to P14.2.
I83	P15.1, P16.1	P15.1 interacts With system axiom P16.1	Type: The action of P15.1 overrides the rule of P16.2. Scenario: The humidity sensor is triggered and P15.1 turns on kitchen fan while. At the same time, occupant uses remote control to switch off kitchen fan. The fan will switch off for a second but then turns on again because the humidity sensor is still triggered. P14.2 cancelled remote control action although occupant wants to switch off the fan. Solution: Assign human control a higher priority.

APPENDIX F: THE IRIS-TS PROTOTYPE DXL CODE

This Appendix presents the DXL code developed for IRIS-TS. The complete DXL code of the tool is more than 70 pages using the format below. Therefore, only the code for executing the first step is presented below to give a feeling of how the DXL code that was written for the IRIS-TS looks like.

```
//////////////////////////////////////////////////////////////////////////////
/***********************************************
 Detecting Requirements Interactions using Semi-Formal
methods IRIS

Copyright © 2004 Mohamed Shehata and Tim Yue.
All rights reserved.
University of Calgary - Canada

Version 3.0
Date: July 20th, 2004
***********************************************/
//////////////////////////////////////////////////////////////////////////////

/***********************************************
This Script will display a welcome Message and determine
whether to proceed or not

***********************************************/

DB graphBox = create ("Welcome to IRIS",
styleFixed|styleFloating|styleCentered)

void repaint(DBE graph) {
   realBackground(graph, realColor_NewGrey4)
   realColor(graph, realColor_Yellow)
   font(graph, 1,1)
   draw(graph, 10,50, "This Program Will Detect
Requirement Interactions Using IRIS")

}

// repaint
DBE graph = canvas(graphBox, 650, 100, repaint)

// Building the Callbacks
void ackHalt(DB graphBox) {
if (confirm("Are you sure to really close?")) {
              release graphBox
              halt
              }
return
}

void proceed(DB graphBox) {
   release graphBox
}

// Adding Buttons
apply(graphBox, "Proceed", proceed)
close(graphBox, true, ackHalt)
block graphBox
/***********************************************
This Script will APPLY THE FIRST STEP OF IRIS WHCIH
IS CLASSIFYING THE REQUIREMENTS INTO SYSTEM
AXIOMS AND DYNAMIC BEHAVIOUR
***********************************************/

/******* INITIALAIZATION**************/
Folder myF = current
Module myM=current
/******** CONSTANTS*****************/
string sRequirementType  = "Classification"
string sRequirementTypeDef = "RequirementType"
string sRequirementAttrType = "RequirementTypeAttr"
string sRequirementAttrEnum[] = { "Dynamic Behaviour",
"System Axiom", "Resource", "N/A" }

/*************Code *********************/
int  iRequirementTypeWidth = 200
string sInitFileName = "/" name(myF) "/" name(myM)
string sFileName

/** Create a dialog box to ask for requirement database **/
DB dbGetFileName = create "Input File Name"

label(dbGetFileName, " Input the file name that contains the
set of requirements to be checked for interactions: \n If you
wish to stop the program and not to proceed press the close
button ")

// DBE dbeFileName = field (dbGetFileName, "File Name: ",
"/Smart Homes/Requirements", 128)

DBE dbeFileName = fileName (dbGetFileName,
sInitFileName);

/** Creates the callback **/
void getFileName(DB dbGetFileName) {
          sFileName = get dbeFileName
          release dbGetFileName
}
void closegetFileNameDB (DB dbGetFileName) {
          release dbGetFileName
          halt
}
```

```
/** Assign buttons to dialog box for requirement database **/
apply (dbGetFileName, "Use", getFileName)
close (dbGetFileName, true, closegetFileNameDB)
block dbGetFileName
Module mCurrent = edit (sFileName, true)
Column cIndex
string sColumnTitle
int n = -1
int loopIndex = 0
Column cReqTypeHandle

/** See if sRequirementAttrType needs to be created **/
AttrType at = find (mCurrent, sRequirementAttrType)
if (at == null) {
        string sErrMessage = ""
        AttrType at = create (sRequirementAttrType,
sRequirementAttrEnum, sErrMessage)
        if (!null sErrMessage) {
                print "Attribute type creation failed!\n"
                halt
        }
}

/** Create attribute "RequirementType" **/
create type sRequirementAttrType attribute
sRequirementTypeDef

/** Loop through all the column to find whether the
Classification column has been created**/
for cIndex in (current Module) do
{
        sColumnTitle = title(cIndex)
        if (matches(sColumnTitle, sRequirementType))
        {
                n = loopIndex
                break
        }
        loopIndex++
}

/** Create Classification column if has not been created **/
if (n == -1)
{
        n = loopIndex
        cReqTypeHandle = insert (column n)
        title (cReqTypeHandle, sRequirementType)
        width (cReqTypeHandle,
iRequirementTypeWidth)
        attribute (cReqTypeHandle,
sRequirementTypeDef)
        save (current Module)
} else {
        cReqTypeHandle = column n
}
refresh (current Module)

/** Creates a dialog box to ask for requirement database **/
DB dbGetSkipToReqName = create "Requirement to skip to"
label(dbGetSkipToReqName, " Enter the name of the
requirement you wish to skip to and press SKIP. If you wish
to start from the begining of the document with NO skipping
press close")
DBE dbeSkipToReqName = field (dbGetSkipToReqName,
"Requirement: ", "Requirement", 128)
string sSkipToReqName = ""
bool bFoundSkipToReqName = false

/***********Creates the callback ***********/
void getSkipToReqName(DB dbGetSkipToReqName) {
        sSkipToReqName = get dbeSkipToReqName
        release dbGetSkipToReqName
}
void closeSkipToReqName (DB dbGetSkipToReqName) {
        bFoundSkipToReqName = true
        release dbGetSkipToReqName
}

/** Assign buttons to dialog box for req. database. **/
apply (dbGetSkipToReqName, "Skip", getSkipToReqName)
close (dbGetSkipToReqName, true, closeSkipToReqName)
block dbGetSkipToReqName

string sReqName
int iRC
Buffer bufTemp = create
Object o
string sClassifyReasons [] = {"Dynamic Behaviour", "System
Axiom", "Resource", "N/A", "Finish"}
Module mnCurrent = edit (sFileName, true)
for o in (current Module) do
{
        sReqName = o."Object Heading"

        bufTemp = sReqName

        if (!bFoundSkipToReqName)
        {
                if (contains (bufTemp,
sSkipToReqName, 0) != 0)
                {
                        continue
                } else
                {
                        bFoundSkipToReqName =
true
                }
        }

/** For each requirement, prompt the user to classify
requirement type **/

DB dbPromptForClassification = create ("Requirements
Classification")
int choice = query (dbPromptForClassification,
"Requirement: \n \n " sReqName " \n \n is Classified as:  ",
sClassifyReasons )
        if (choice == 4)
        {
          break;
        } else if (choice == 0)
        {
          o.sRequirementTypeDef = "Dynamic
Behaviour"
        } else if (choice == 1)
        {
          o.sRequirementTypeDef = "System Axiom"
        } else if (choice == 2)
        {
          o.sRequirementTypeDef = "Resource"
        } else if (choice == 3)
        {
          o.sRequirementTypeDef = "N/A"
        }

}
```

```
if (!bFoundSkipToReqName)
            warningBox "String not found!"

/** Defintions for defining the columns **/
n = -1
loopIndex = 0

/** Loop through all the columns to find whether
classificationColumn is created **/
for cIndex in (current Module) do
{
            sColumnTitle = title (cIndex)
            if (sColumnTitle == "Classification")
            {
                        n = loopIndex
                        break;
            }
            loopIndex++
}


/** Create the ClassificationColumn if one has not been
created. **/
if (n == -1)
{
      n = loopIndex
      Column classificationColumn = insert(column n)
       title (classificationColumn, "Classification")
       width (classificationColumn, 200)
        attribute (classificationColumn, "RequirementType")
        save (current Module)
}
refresh (current Module)

/* Confirm if user want to proceed to IRIS step 2 or stop */

DB goStepTwoBox = create ("Proceed to Step 2 of IRIS",
styleFixed|styleFloating|styleCentered)

void repaint1(DBE goStepTwo) {
   realBackground(goStepTwo, realColor_NewGrey4)
   realColor(goStepTwo, realColor_Yellow)
   font(goStepTwo, 1,1)
   draw(goStepTwo, 10,30, "Do you wish to proceed to step
2 of IRIS:")
   draw(goStepTwo, 10,60, "Requirements Attributes
Idenitifcation")
}

// repaint
DBE goStepTwo = canvas(goStepTwoBox, 450, 80,
repaint1)

// Building the Callbacks

void ackHalt1(DB goStepTwoBox) {
if (confirm("Are you sure to really Exit?")) {
            release goStepTwoBox
            halt
            }
return
}

void proceed1(DB goStepTwoBox) {
    release goStepTwoBox
}

// Adding Buttons
apply(goStepTwoBox, "Proceed", proceed1)
close(goStepTwoBox, true, ackHalt1)
block goStepTwoBox

////////////////////////////////////////////////////////////////////
/****************************************
              End of Step 1 of IRIS

****************************************/
////////////////////////////////////////////////////////////////////
```