2019

Situation-oriented requirements engineering

Nimanthi Atukorala

Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/etd

Part of the Computer Sciences Commons

Recommended Citation

Atukorala, Nimanthi, "Situation-oriented requirements engineering" (2019). Graduate Theses and Dissertations. 16965.
https://lib.dr.iastate.edu/etd/16965

This Dissertation is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
Situation-oriented requirements engineering

by

Nimanthi L. Atukorala

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Science

Program of Study Committee:
Carl K. Chang, Major Professor
    Samik Basu
    Ying Cai
    Simanta Mitra
    Johnny Wong

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this dissertation. The Graduate College will ensure this dissertation is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2019

Copyright © Nimanthi L. Atukorala, 2019. All rights reserved.
DEDICATION

To Appachchi, Amma, Rumesh and Sanuki,

thank you for everything.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>vi</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>vii</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>ix</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>x</td>
</tr>
<tr>
<td>CHAPTER 1. OVERVIEW</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Introduction to Requirements Engineering</td>
<td>1</td>
</tr>
<tr>
<td>1.1.1 Definition of Requirements Engineering</td>
<td>1</td>
</tr>
<tr>
<td>1.1.2 Overall Requirements Engineering procedure</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Situation-oriented Requirements Engineering: Proposed approach</td>
<td>4</td>
</tr>
<tr>
<td>1.2.1 Research problem</td>
<td>4</td>
</tr>
<tr>
<td>1.2.2 Overview of the proposed approach</td>
<td>5</td>
</tr>
<tr>
<td>1.3 Dissertation organization</td>
<td>7</td>
</tr>
<tr>
<td>CHAPTER 2. REVIEW OF LITERATURE</td>
<td>8</td>
</tr>
<tr>
<td>2.1 Early phase</td>
<td>8</td>
</tr>
<tr>
<td>2.2 Goal-Oriented Requirements Engineering</td>
<td>9</td>
</tr>
<tr>
<td>2.3 Agent-Oriented Requirements Engineering</td>
<td>12</td>
</tr>
<tr>
<td>CHAPTER 3. SITUATION-TRANSITION STRUCTURE</td>
<td>14</td>
</tr>
<tr>
<td>3.1 Situation-transition structure generating procedure</td>
<td>14</td>
</tr>
<tr>
<td>3.1.1 Choosing the significant human subjects and the domain</td>
<td>15</td>
</tr>
<tr>
<td>3.1.2 Collecting raw data in an Internet of Things environment</td>
<td>15</td>
</tr>
<tr>
<td>3.1.3 Identifying behavioral and environmental context components</td>
<td>17</td>
</tr>
<tr>
<td>3.1.4 Pre-processing raw data</td>
<td>19</td>
</tr>
<tr>
<td>3.1.5 Selecting the human desires of interest and prioritization of behavioral context components states for each desire</td>
<td>20</td>
</tr>
<tr>
<td>3.1.6 Generating sequence of situations</td>
<td>21</td>
</tr>
<tr>
<td>3.1.7 Deriving situation-transition structure</td>
<td>26</td>
</tr>
<tr>
<td>3.1.8 Mutual information values</td>
<td>31</td>
</tr>
<tr>
<td>CHAPTER 4. SITUATION-ORIENTED REQUIREMENTS ELICITATION</td>
<td>32</td>
</tr>
<tr>
<td>4.1 Analyzing situation-transition structure</td>
<td>32</td>
</tr>
<tr>
<td>4.1.1 Rooted situation-transition structure</td>
<td>32</td>
</tr>
<tr>
<td>4.1.2 Definitions of property extracted terms</td>
<td>32</td>
</tr>
<tr>
<td>4.2 New requirements elicitation using situation-transition structure</td>
<td>34</td>
</tr>
<tr>
<td>4.2.1 Definitions of requirements construction terms</td>
<td>34</td>
</tr>
<tr>
<td>4.3 Case Studies</td>
<td>34</td>
</tr>
<tr>
<td>4.3.1 Case Study 1: Smart home energy management system</td>
<td>35</td>
</tr>
<tr>
<td>4.3.2 Case Study 2: Smart personal assistant</td>
<td>36</td>
</tr>
</tbody>
</table>
4.3.3 Case Study 3: Cooperative research environment ........................................ 40
4.4 Discussion of proposed situation-oriented requirements elicitation approach ...... 46
4.5 Literature on requirements elicitation ................................................................. 48
  4.5.1 Observation approach in requirements elicitation ........................................ 49
  4.5.2 Situation awareness in requirements elicitation ............................................ 49
  4.5.3 Individualized requirements elicitation ......................................................... 50
  4.5.4 Crowdsourcing and requirements elicitation ................................................ 50
  4.5.5 Contextual requirements ............................................................................ 51
  4.5.6 Activity theory and requirements elicitation ................................................ 51

CHAPTER 5. SITUATION-ORIENTED REQUIREMENTS RISK MANAGEMENT ................................................................. 53
  5.1 Human-centered risk management ................................................................. 53
  5.2 Features and feature models ........................................................................ 54
  5.3 Situation-oriented human-centered requirements risk management .......... 57
    5.3.1 Deriving situation-transition structure from observational data .......... 57
    5.3.2 Eliciting set of requirements of the software-to-be ............................. 58
    5.3.3 Deriving the initial set of features of the software-to-be ..................... 59
    5.3.4 Generating the feature model of the software-to-be ......................... 59
    5.3.4.1 Feature model decomposition ......................................................... 59
    5.3.4.2 Feature models composition .......................................................... 67
    5.3.5 Risk management of elicited requirements .......................................... 68
      5.3.5.1 Risk identification ................................................................. 71
      5.3.5.2 Risk assessment ................................................................. 73
      5.3.5.3 Risk planning and monitoring ................................................. 73
  5.4 Case Study .................................................................................................. 76
  5.5 Discussion of proposed situation-oriented human-centered requirements risk management approach ................................................................. 93
  5.6 Literature on requirements risk management ................................................. 95
    5.6.1 Common risk assessment tools and techniques .................................. 95
      5.6.1.1 Preliminary hazard analysis ...................................................... 95
      5.6.1.2 Failure mode and effect analysis ............................................ 96
      5.6.1.3 Fault tree analysis ................................................................. 96
      5.6.1.4 Hazards and operability analysis ............................................ 96
    5.6.2 Probabilistic risk assessment and human reliability analysis ............ 97
    5.6.3 Risk matrix ......................................................................................... 98
    5.6.4 Goal-oriented risk assessment approaches ..................................... 99
    5.6.5 Machine learning based risk assessments ....................................... 100
    5.6.6 Risk assessment in continuously improving and changing software development ................................................................. 102

CHAPTER 6. SITUATION-ORIENTED REQUIREMENTS EVALUATION AND PRIORITIZATION ................................................................. 103
  6.1 Human factors in requirements prioritization ............................................ 103
  6.2 System factors in requirements prioritization ............................................ 104
  6.3 Risk factor of requirements ....................................................................... 105
  6.4 Requirements prioritization ....................................................................... 106
**LIST OF FIGURES**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 3.1</td>
<td>Overall process of proposed situation-transition structure generation</td>
<td>14</td>
</tr>
<tr>
<td>Figure 3.2</td>
<td>Various methods of collecting raw observational data in the IoT setting</td>
<td>16</td>
</tr>
<tr>
<td>Figure 3.3</td>
<td>Two categories of raw data collecting components and their possible states</td>
<td>18</td>
</tr>
<tr>
<td>Figure 3.4</td>
<td>Sample of preprocessed sequence of situations</td>
<td>26</td>
</tr>
<tr>
<td>Figure 3.5</td>
<td>Sample of situation-transition structure</td>
<td>27</td>
</tr>
<tr>
<td>Figure 4.1</td>
<td>Set of possible paths in situation-transition structure in Figure 3.5</td>
<td>33</td>
</tr>
<tr>
<td>Figure 4.2</td>
<td>Part of the situation-transition structure of Smart* dataset</td>
<td>38</td>
</tr>
<tr>
<td>Figure 4.3</td>
<td>Part of the situation-transition structure of MIT dataset</td>
<td>41</td>
</tr>
<tr>
<td>Figure 4.4</td>
<td>Some Cooperative Research Environment (CoRE) website interfaces</td>
<td>43</td>
</tr>
<tr>
<td>Figure 4.5</td>
<td>Part of the situation-transition structure of CoRE dataset</td>
<td>44</td>
</tr>
<tr>
<td>Figure 5.1</td>
<td>Sample of feature model</td>
<td>56</td>
</tr>
<tr>
<td>Figure 5.2</td>
<td>Overall process of proposed Situation-oriented requirements risk management</td>
<td>58</td>
</tr>
<tr>
<td>Figure 5.3</td>
<td>Sample of derived feature model from Algorithm 5.1</td>
<td>67</td>
</tr>
<tr>
<td>Figure 5.4</td>
<td>Case study 4.3.2 Require requirement feature model</td>
<td>77</td>
</tr>
<tr>
<td>Figure 5.5</td>
<td>Case study 4.3.2 Achieve requirement feature model</td>
<td>80</td>
</tr>
<tr>
<td>Figure 5.6</td>
<td>Case study 4.3.2 Maintain requirement feature model</td>
<td>82</td>
</tr>
<tr>
<td>Figure 5.7</td>
<td>Case study 4.3.2 Stop requirement feature model</td>
<td>85</td>
</tr>
<tr>
<td>Figure 5.8</td>
<td>Case study 4.3.2 Avoid requirement feature model</td>
<td>88</td>
</tr>
<tr>
<td>Figure 5.9</td>
<td>Smart personal assistant overall feature model</td>
<td>91</td>
</tr>
<tr>
<td>Figure 6.1</td>
<td>Iterative and incremental software development process</td>
<td>107</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 4.1</td>
<td>Definitions of property extracted terms</td>
<td>33</td>
</tr>
<tr>
<td>Table 4.2</td>
<td>Set of requirements construction terms</td>
<td>35</td>
</tr>
<tr>
<td>Table 4.3</td>
<td>Sample set of requirements elicited from situation-transition structure in Figure 4.2</td>
<td>39</td>
</tr>
<tr>
<td>Table 4.4</td>
<td>Sample set of requirements elicited from situation-transition structure in Figure 4.3</td>
<td>42</td>
</tr>
<tr>
<td>Table 4.5</td>
<td>Sample set of requirements elicited from situation-transition structure in Figure 4.5</td>
<td>45</td>
</tr>
<tr>
<td>Table 5.1</td>
<td>Basic feature model notations</td>
<td>57</td>
</tr>
<tr>
<td>Table 5.2</td>
<td>Case study 4.3.2 Require requirement feature model details</td>
<td>78</td>
</tr>
<tr>
<td>Table 5.3</td>
<td>Case study 4.3.2 Require requirement feature model details (cont.)</td>
<td>79</td>
</tr>
<tr>
<td>Table 5.4</td>
<td>Case study 4.3.2 Achieve requirement feature model details</td>
<td>81</td>
</tr>
<tr>
<td>Table 5.5</td>
<td>Case study 4.3.2 Maintain requirement feature model details</td>
<td>83</td>
</tr>
<tr>
<td>Table 5.6</td>
<td>Case study 4.3.2 Maintain requirement feature model details (cont.)</td>
<td>84</td>
</tr>
<tr>
<td>Table 5.7</td>
<td>Case study 4.3.2 Stop requirement feature model details</td>
<td>86</td>
</tr>
<tr>
<td>Table 5.8</td>
<td>Case study 4.3.2 Stop requirement feature model details (cont.)</td>
<td>87</td>
</tr>
<tr>
<td>Table 5.9</td>
<td>Case study 4.3.2 Avoid requirement feature model details</td>
<td>89</td>
</tr>
<tr>
<td>Table 5.10</td>
<td>Case study 4.3.2 Avoid requirement feature model details (cont.)</td>
<td>90</td>
</tr>
<tr>
<td>Table 5.11</td>
<td>Smart personal assistant software risk identification</td>
<td>92</td>
</tr>
<tr>
<td>Table 5.12</td>
<td>Smart personal assistant requirements qualitative risk assessment</td>
<td>93</td>
</tr>
<tr>
<td>Table 6.1</td>
<td>Smart personal assistant requirements human importance factor</td>
<td>109</td>
</tr>
<tr>
<td>Table 6.2</td>
<td>Smart personal assistant requirements system importance factor</td>
<td>110</td>
</tr>
<tr>
<td>Table 6.3</td>
<td>Smart personal assistant requirements risk factor</td>
<td>111</td>
</tr>
</tbody>
</table>
Table 6.4  *Smart personal assistant requirements prioritization* .................................. 111
ACKNOWLEDGMENTS

First and foremost, I would like to express my sincere gratitude to my advisor Professor Carl K. Chang, for his continuous support, patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this dissertation. He has taught me, both consciously and unconsciously, how to be a good researcher as well as a better human being. It has been an honor to be one of his Ph.D. students.

Besides my advisor, I would like to thank the rest of my program of study committee members: Dr. Johnny Wong, Dr. Samik Basu, Dr. Ying Cai, and Dr. Simanta Mitra, for their insightful comments and encouragement. I thank all former and current lab mates in Software Engineering Lab and Computer Science Department staff for their suggestions and helpful collaboration.

Last but not least, I would like to thank my father, Nandasoma Atukorala, my mother, Lalani Dhanasekara, my husband, Rumesh Piyasinghe and our little angel Sanuki Piyasinghe for the unconditional love and support. You gave me the strength and determination to complete the Ph.D.
ABSTRACT

The establishment of smart environments, Internet of Things (IoT) and socio-technical systems has introduced many challenges to the software development process. One such main challenge is software requirements gathering which needs to address issues in a broader spectrum than traditional standalone software development. Consideration of bigger picture that includes software, its domain, the components of the domains and especially the interactions between the software and the surrounding domain components, including both human and other systems entities, is essential to gathering reliable requirements. However, most of the traditional Requirements Engineering approaches lack such comprehensive overlook of the overall view.

The main objective of this work is to introduce a human-centered approach to Requirements Engineering in order to push the boundaries of traditional concepts to be more suitable for use in the development of modern socio-technical systems in smart environments. A major challenge of introducing a human-centered approach is to effectively identify the related human factors; especially, since each individual has unique desires, goals, behaviors. Our proposed solution is to use the observational data sets generated by smart environments as a resource to extract individual’s unique personalities and behaviors related to the software design. The concept of situations defined in our earlier study is used to represent the human and domain related aspects including human desires, goals, beliefs, interactions with the system and the constrained environment.

In the first stage of this work, a computational model called situation-transition structure is developed to understand the discrete factors and behavior patterns of
individuals through the observational data. During the second stage, the information mined from the situation-transition structure is applied to propose new human-centered approaches to support main Requirements Engineering concepts: requirements elicitation, risk management, and prioritization. The pertinence of the proposed work is illustrated through some case studies. The conclusion asserts some of the future research direction.

**Keywords:** human-centered, Internet of Things (IoT), Requirements Engineering, situation, situation-transition structure, smart environments, socio-technical systems
CHAPTER 1. OVERVIEW

In this chapter, we provide an overview of our proposed situation-oriented, human-centered approach to Requirements Engineering.

1.1 Introduction to Requirements Engineering

Requirements Engineering in the context of Software Engineering deals with elicitation, refinement, analysis, verification, and validation of software system requirements. From the past experience, it has been accepted that inadequate, incomplete, ambiguous or inconsistent requirements have a significant impact on the quality of the software system [1]. However, identifying the correct requirements is challenging [2] and no standard procedure exists. Hence, the area of Requirements Engineering gained a lot of attention in academia as well as in the industry during the last two decades.

1.1.1 Definition of Requirements Engineering

Definition of “requirements” by Ross et al. [3] is considered as one of the oldest definitions of Requirements Engineering and frequently used by authors as a pioneering work.

“Requirements definition is a careful assessment of the need that a system is to fulfill. It must say why a system is needed, based on current or foreseen conditions, which may be internal operations or external market. It must say what system features will serve and satisfy this context. And it must say how the system is to be constructed” [3].

According to the definition by Ross et al. [3], the major concern of Requirements Engineering must be to provide a clear description about the purpose of the software system,
functionalities to be included to the software system to achieve that purpose and the possible constraints on designing and implementing such a software system.

At present, many alternative definitions of Requirements Engineering can be found in the literature as a result of changing the focus of the Requirements Engineering into different directions with time. For example, in [4] the Requirements Engineering is defined as “the process of discovering the purpose of a software system by identifying stakeholders and their needs and by documenting these in a form that is amenable to analysis, communication, and subsequent implementation”. Here the term “stakeholder” represents the group of people or organizations who will be affected by the software system and who have a direct or indirect influence on the system requirements [5]. As such, a broader enterprise perspective began to emerge.

Similarly, [6] defined Requirements Engineering as “the branch of Software Engineering concerned with the real-world goals for, functions of, and constraints on the software system. It is also concerned with the relationship of these factors to precise specifications of software behavior, and their evolution over time and across software families”. Thus, formal specification plays an increasingly important role in Requirements Engineering.

1.1.2 Overall Requirements Engineering procedure

Today, various Requirements Engineering procedures are used in practice [1][4][7]. However, a general belief is that the Requirements Engineering is an iterative, multi-step procedure. Moreover, most of the existing procedures share the following steps:
- **Domain analysis:** Studying the environment for the system-to-be is known as the domain analysis. In this phase, the relevant stakeholders are identified and interviewed. Further, the problems with the existing system are discovered and possibilities for improvement are investigated. Objectives for the target system are also identified.

- **Elicitation:** During the elicitation phase the alternative models for the target system are analyzed to meet the identified objectives. Requirements and assumptions on components of such models are identified.

- **Negotiation and agreement:** In this phase the alternative requirements and assumptions are evaluated and the risks are analyzed. Next, the set of requirements to be included in the system-to-be is selected.

- **Specification and specification analysis:** Here, the requirements and assumptions are formulated precisely. Then, the specifications are tested for problems such as incompleteness, inconsistency, feasibility, etc.

- **Documentation:** Decisions made during the Requirements Engineering process are documented together with the underlying rationale and assumptions during this phase.

- **Evolution:** In the evolution phase, current requirements are modified to accommodate corrections, environmental changes, or new objectives [1].
It is accepted that applying these steps iteratively could lead to a well-defined set of requirements.

1.2 Situation-oriented Requirements Engineering: Proposed approach

1.2.1 Research problem

In the era of smart environments, Internet of Things (IoT) and socio-technical systems, the development of reliable software implies the need of comprehensive and computational Requirements Engineering approaches that can successfully gather requirements by analyzing the overall view which includes the software, its domain, the other components of the domains and the interactions between the software and the surrounding domain components that enclose both human and other systems entities. However, most of the traditional Requirements Engineering approaches lack such inclusive analysis capability leading the requirements engineers to gather an incomplete, inconsistent set of requirements which results in a negative impact on the overall software development [8].

Among various other components existing in the domain of the software, we believe that the human component is one of the most complex and challenging aspects that directly affect the success of the software. Especially, the end-users of the software play an important role in Requirements Engineering. It is important to note that each individual end-user is unique and has his or her own unique interests, desires, personal goals, etc. [9]. Moreover, their origin, culture and surrounding environmental factors lead them to have their own unique beliefs. In this work, our first objective is to ease the limitations in traditional requirements elicitation approaches by introducing a human-centered approach to
requirements elicitation which considers the end-users’ human factors and the surrounding environmental factors.

Requirements prioritization is an inevitable process in modern software development. We believe that requirements prioritization in socio-technical systems must balance between human and system viewpoints and favor more important requirements. Moreover, as our work is mainly focused on the end-users, we believe that requirements prioritization must also pay special attention to the consequences of the interactions between end-users and the software. Hence, our second objective is to introduce a requirements risk management approach to minimize the negative impact involved with the interactions between end-users and the software. The third objective of this work is to use the outcomes of requirements risk management along with the information gathered during the requirements elicitation to define our human-centered requirements prioritization approach.

1.2.2 Overview of the proposed approach

Our proposed work includes two main stages. In the first stage, we developed a computational model to understand the discrete human factors and behavior patterns of end-users through the collected time-stamped observational data. We believe that the end-users’ individual human factors are directly connected to their goals and desires; or in general, their motivational mental state and the factors in the surrounding environment [10]. In order to configure the human mental state and environmental factors into a computational model, the proposed method borrows the concept of situations presented in an earlier study [10]. The term “situation” refers to a clear computational unit, which by this peculiar definition includes the human mental state, behavioral and environmental contexts for a predefined time period. Although a human mental state is not visible, in an earlier study, it is believed
that the mental state can be predicted to some extent through monitoring the human activities and the environment [11][12]. With this definition of the situation, it is possible to pair time-stamped records of desire, set of actions and environmental context values of a particular person into situation tuples which leads to a sequence of situations with time. One significant observed property in the sequence of situations of an individual is that some situations are more likely to transition to a specific subset of future situations than others. We call this property as “situation transition”. Hence, we define the term “situation-transition structure” as a domain-specific directed weighted graph generated using all possible situation transitions of an individual in a particular domain during a pre-defined time period.

In the second stage, we applied the information extracted from the situation-transition structure to enhance the prevalent Requirements Engineering methods by including the concept of situation for requirements elicitation, risk management, and prioritization.

The major contributions of this work include,

1. Introducing a novel machine learning based approach to derive situation-transition structure which graphically represents the human behavioral patterns and the environmental context of a given domain.

2. Introducing a novel human-centered approach to requirements elicitation based on situation-transition structure.

3. Introducing a novel human-centered approach to requirements risk management which effectively considers the risks involved in human-system interactions.
4. Introducing a novel human-centered approach to requirements evaluation and prioritization which can be extended to the iterative and incremental software development process models.

1.3 Dissertation organization

The rest of this dissertation is organized as follows: Chapter 2 reviews the literature on existing approaches in Requirements Engineering. Chapter 3 summarizes the concept of situations from our previous study and provides a detailed step-by-step procedure of generating the situation-transition structure. Chapter 4 describes the proposed requirements elicitation using three situation-transition structures from a real-life datasets. Chapter 5 presents our proposed requirements risk management approach including a case study. In Chapter 6, the proposed requirements evaluation and prioritization procedure is given. Finally, Chapter 7 concludes the dissertation with a future research road map.
CHAPTER 2. REVIEW OF LITERATURE

This chapter reviews the evolution of Requirements Engineering and provides a brief introduction to the most common approaches used in Requirements Engineering in recent years.

2.1 Early phase

Although the idea of requirements had been known for quite some time (e.g., [3]), many aspects of Requirements Engineering were considered as parts of the designing phase until the mid-1980s [13]. In the 1990s, several researchers helped to establish some important properties of software requirements. Eventually software engineers began to treat Requirements Engineering as a separate domain with its own rights [7][13].

In [14] Michael Jackson proposed the distinction between “machine” and “environment”. The “machine” was defined as “one or more computers that behave in a way to satisfy the requirements with the help of software” whereas the “environment” was defined as “the part of the world that the machine will interact and in which the effects of the machine will be observed”. Moreover, he stated that a machine installed in an environment can influence the environment or be influenced by the environment only if there are common events and states (common phenomena) for both the machine and the environment. Hence, requirements are located in the environment and can be represented as some conditions over the events and states of the environment. In other words, it is possible to formulate and state the requirements in a stakeholder friendly manner without referring to the machine.

Both Jackson [14] and Parnas [15] independently made the distinction between “requirements” and “domain properties”. According to Jackson [14], a machine may
indirectly affect the private phenomena of the environment due to the causal chains in the environmental phenomena. In order to avoid such influence, the description of the requirements must include both desired (optative) conditions over the environment phenomena (as requirements) and given (indicative) properties of the environment (as domain properties).

Jackson [14] and Parnas [15] also made the distinction between the “requirements” and the “software specifications”. They proposed that the software specification must be expressed in terms of the machine phenomena in the language accessible to software developers. In addition, both “requirements” and the “environmental assumptions” (also known as “expectations”) are optative; however, requirements enforced by the software, whereas the assumptions enforced by other components in the environment. Assumptions specify what the system expects from the environment.

The establishment of Requirements Engineering as a separate domain and the idea of representing requirements in stakeholder perspective propelled Requirements Engineering researchers to introduce new approaches. Rest of this chapter provides a brief introduction to two such widely known approaches: Goal-Oriented Requirements Engineering (GORE) and Agent-Oriented Requirements Engineering (AORE).

### 2.2 Goal-Oriented Requirements Engineering

The idea of the goal is used in most of the recently proposed Requirements Engineering methods and techniques. According to the current Requirements Engineering literature, the term “goal” has a number of definitions. For example, van Lamsweerde [7] defines a goal as “an objective that the system should achieve through the cooperation of agents in the software-to-be and in the environment”. Here the term “agent” is referred to
the active components in the system-to-be and its environment. Anton [16] states that “goals are high-level objectives of the business, organization or system; they capture the reasons why a system is needed and guide decisions at various levels within the enterprise”.

In Goal-Oriented Requirements Engineering (GORE) the “agents” are responsible for achieving goals. Hence, a “requirement” is defined as “a goal whose achievement is the responsibility of a single software agent” whereas an “assumption” is defined as “a goal whose achievement is delegated to a single agent in the environment” [7].

**Benefits of Goal-Oriented Requirements Engineering**

There are a number of important benefits associated with explicit modeling, refinement, and analysis of goals [17].

- **Added support for the early requirements analysis**: GORE provides information on a broader System Engineering spectrum compared to the traditional Requirements Engineering methods. Goals represent the rigorous assertions expected from the system made of both software-to-be and its environment. Hence, the software requirements specifications include clear explanations of domain properties and expectations about the environment. Moreover, the goals also provide a rationale for requirements.

- **Goals provide a precise criterion for completeness of a requirements specification.** The specification is complete with respect to a set of goals if all the goals can be proven to be achieved from the specification and the properties known about the domain [18].

- **Goals provide a precise criterion for requirements pertinence.** A requirement is pertinent with respect to a set of goals in the domain if its specification is used in the proof of one goal at least [18].
• A goal refinement tree can be used to determine the traceability links from high-level strategic system objectives to low-level technical requirements.

• Goal modeling provides a natural mechanism for structuring complex requirements documents [17].

• Goals can be used to provide the basis for the detection and management of requirements conflicts [19][20].

• Alternative solutions in the problem domain can be detected and analyzed using the goal model by considering the alternative goals and assignment of responsibilities.

• Goal models simplify the communication between Requirements Engineers and customers. It provides the right level of abstraction to the decision makers to validate the selected choices and to suggest another alternative.

• It is believed that the goals are much more stable than requirements [16][17]. A requirement represents one particular way of achieving a goal and therefore, the requirement is more likely to evolve towards a different way of achieving that same goal than the goal itself. In general, a goal is more stable if it is at a higher level.

**Limitations of Goal-Oriented Requirements Engineering**

• Identifying goals is not an easy task. Goals could be explicitly stated by the stakeholders or in the various sources of information available to requirements engineers. However, most frequently, goals are implicit and therefore the elicitation process must take place.

• Goals are not human-centered. Although the approach includes the domain properties and expectations of the environment, it does not explicitly handle the human-related
information. As mentioned earlier goals represent the assertions expected from the system, but not express the assertions from a human perspective.

### 2.3 Agent-Oriented Requirements Engineering

The underline belief of the Agent-Oriented Requirements Engineering (AORE) is that the Requirements Engineering must mainly focus on the representation of the intended system in relation to its environment [21][22]. According to Yu [22], the term “agent” represents “any active element in the environment including the target system”. Although the AORE is applicable during Requirements Engineering of any software development project, it is mainly suggested for the distributed system design [22].

**Benefits of Agent-Oriented Requirements Engineering**

Agent Orientation brings several important benefits [23].

- The key benefit of using the concept of “agents” as a guiding concept during Requirements Engineering is that it will bring the issues centered on an agent together so that they can be identified and addressed.

- The specification of agents in terms of their mental attitudes (beliefs, goals, commitments, etc.) provides a higher level of abstraction.

- Assignment of mental attitudes to agents could provide the ability to explain or predict the agents’ behavior even with little information about their internal structure. Moreover, the mental attributes also play a major role in understanding the changing behaviors of agents with respect to the changes in their environment or organization.
During the modeling of system’s operational organization or environment, the communication between agents can be represented in abstract level as various types of “speech acts” without specifying the form and the mechanism of messages. Moreover, it is possible to import the analyses of multi-agent cooperative problem-solving models. Social relations and social rules can also be integrated into such models.

Requirements Engineering tools can be promoted by the implementation techniques for agent-oriented frameworks so that those tools can provide more powerful and effective modeling and analysis techniques.

Limitations of Agent-Oriented Requirements Engineering

The end-user is considered as a regular agent in the environment and the interactions between agents are determined through the system perspective. In other words, the target system becomes the centering agent of AORE.
CHAPTER 3. SITUATION-TRANSITION STRUCTURE

In this chapter, we describe the situation-transition structure generating procedure in detailed.

3.1 Situation-transition structure generating procedure

Figure 3.1 represents the overall process of the proposed method. The process starts with (1) identifying relevant human subjects, the domain of interest and collecting initial raw data by observing regular activities of those human subjects in the domain. Next, (2) the collected raw data are pre-processed in order to extract the behavioral and environmental context with time. (3) The definition of the situation is then employed to encapsulate the data into more computationally rich information units which can be used to (4) derive a sequence of situations with time. Finally, (5) the situation-transition structure is generated by applying machine learning based computational model to the derived situation sequence. Note that the generated situation-transition structure can be used to extract information on human interest, believes, lifestyle and even existing environmental or external constraints.

Figure 3.1 Overall process of proposed situation-transition structure generation
3.1.1 Choosing the significant human subjects and the domain

Selecting the relevant human subjects and the domain is the first stage of generating the situation-transition structure. The selection of human subjects depends on the primary goal of the situation-transition structure generation. The term “domain” implies the environment where the situations of interest are most likely to occur. Selection of domain also depends on the primary goal of the situation-transition structure generation as well as the particular human subjects.

In Software Engineering the human subjects are called stakeholders that refer to any individuals or groups or organization that may affect, affected by the developed software. These stakeholders can be divided into two main categories as development team members and the end-users. Although the individual human factors of development team directly affect the effectiveness of the software development process, the final decision on successfulness depends on the end-users satisfaction. Hence, we are focusing on end-users as human subjects in generating the situation-transition structure for software development applications.

Note that the selected set of human subjects and the boundaries of the domain can be updated during the process but may increase the time duration of the entire process and resource consumption.

3.1.2 Collecting raw data in an Internet of Things environment

As we have entered the Internet of Things (IoT) era, requirements engineers can often collect a lot of raw data from such a sensor-laden environment. Thus, raw data can be collected through observation of both the regular activities of the human subjects and the relevant environmental factors in the domain. The indirect observation can be performed
through data collecting component such as sensors. We assume that each data collecting component is uniquely labeled as ComponentType:Location:ID and the data collected from these components are recorded with time.

Some possible approaches to observe the activities of the human subjects and the environmental factors are given in Figure 3.2. Note that humans are also a kind of sensor.

![Figure 3.2 Various methods of collecting raw observational data in the IoT setting](image)

The observation is conducted without any kind of interaction with the participants, assuming that prior agreement has been reached and the participants know that they are being observed. Further, there are no particular activities that are planned to observe in advance. All activities were given equal priority. Therefore, according to Robson [24], the observation in the proposed method can be classified as an unobtrusive and unstructured approach. We believe that these properties increase the flexibility and, thus the effectiveness of the proposed method. Moreover, the duration of data collection can be decided by domain expert based on the human subjects and the domain of interest.
3.1.3 Identifying behavioral and environmental context components

Raw data collecting components can be divided into two categories as Behavioral Context Components (BCC) and Environmental Context Components (ECC). Components that provide data about activities such as motion belong to BCC, whereas components that provide data about environmental factors such as location, temperature belong to ECC.

We assume that the states of ECC can be in two forms as discrete states and continuous states. For example, the state of ECC that records location can be one of the possible locations defined using the coordinate values, however, the state of ECC that records temperature can be laid in the range $-30.0^\circ F$ to $120.0^\circ F$. Note that in either case, ECC does not imply any activity been occurred or not.

Conversely, we assume that all the BCC have set of known discrete states based on whether it “sensed a signal” or not, and can be recorded with time whenever their states changed from one to another. The states remain the same during the time intervals between two twists of states. Note that the time intervals between two twists are not regular. For example, the state of a BCC use for motion detection can be either ON or OFF and the twist from ON state to OFF state at time $t_1$ and OFF state to ON state at time $t_2$ can be recorded along with time $t_1$ and $t_2$. Then, the BCC state remains OFF from $t_1$ to $t_2$ period of time. Moreover, we assume that all possible states of BCC can be categorized as “positive states” and “negative states” (Figure 3.3).

**Definition:** Positive states of a BCC are the set of states to represent that BCC “sensed a signal” or the states that indicate “activity had occurred”. Negative states of a BCC are the set of states to represent that BCC “did not sense a signal” or the states that indicate “activity had not been occurred”.
Analysts or domain experts can select the positive and negative states for BCC based on their type and functionality. However, differentiating positive and negative states could be difficult in cases where data collecting components engaged in multiple activities. The selection of positive and negative BCC can be considered in two cases.

Let $S$ be the set of all possible states of a single behavioral context component.

- **Case 1:** If all states in $S$ correspond to a single activity, then let $P \subseteq S$ represents the particular activity had occurred and the complement of $P$ (say $P'$) represents the activity had not occurred. Then, select $P$ as positive states set and $P'$ as negative states set. If $P'$ is empty, introduce a fictional state as NULL for the negative state. For example, if a motion detection sensor has two states, $S=\{ON, OFF\}$, and the ON state represents a “movement had occurred in a particular area” whereas OFF state represents “no movement occurred in a particular area”, then, the subset $P=\{ON\}$ becomes the positive state, while $P'=\{OFF\}$ becomes the negative state. Another example is if the two states $S=\{OPEN, CLOSE\}$ of a door-closure sensor represent the same activity, then
$P = S = \{\text{OPEN, CLOSE}\}$ becomes the positive states and $P' = \{\text{NULL}\}$ becomes the negative state which is fictional.

- **Case 2:** If states in $S$ correspond to different activities, then consider that component as a group of separate components where each component $i$ in the group has a subset $P_i$ of $S$ to denote that the same activity $a_i$ had occurred. All the remaining states in $S$, i.e. the complement of $P_i$ (say $P_i'$), denote that $a_i$ had not occurred. Again, select $P_i$ as positive states set and $P_i'$ as negative states set of component $i$. If $P_i'$ is empty, introduce a fictional state. For example, if $S = \{\text{ON, OFF}\}$ for a switch-state detection sensor represents three different activities $X$, $Y$ and $Z$, such that $\text{ON}$ state represents either $X$ or $Z$ had occurred and $\text{OFF}$ state represents either $Y$ or $Z$ had occurred, then that sensor can be regarded as a group of three sensors with positive states set $P_x = \{\text{ON}\}$, $P_y = \{\text{OFF}\}$, $P_z = \{\text{ON, OFF}\}$, and negative states set $P_x' = \{\text{OFF}\}$, $P_y' = \{\text{ON}\}$, $P_z' = \{\text{NULL}\}$, respectively.

### 3.1.4 Pre-processing raw data

During pre-processing of raw data, we discretize the time into small equal-width intervals, and the state of each BCC is recorded at the end of each time interval. All BCC that do not have positive states at these time instances will be recorded as in their negative states. The analysts can determine the width of the time interval. Reliability of the process will increase for smaller time intervals. In addition, states of each ECC at these instances are recorded separately. If a state of ECC varies within the time interval, then the average value can be computed and recorded.
The difference between the recorded times of the previous record and the current record is called the time interval of the current record. Our methodology assumes that the BCC states are constant within this time interval. Amount of information lost due to this assumption can be reduced by selecting smaller time intervals.

3.1.5 Selecting the human desires of interest and prioritization of behavioral context components states for each desire

Next step in the proposed method is to select a sample of desires that have influences on the system-to-be. This step needs to be accomplished by analysts with prior knowledge on human subjects, domains of interest, and the system-to-be. Quick interviews with participants can be helpful during this process. The granularity of the desires depends on human subjects, the domain of interest and the system-to-be. After selecting the human desires of interest, the analysts need to provide a priority BCC states set for each desire.

**Definition:** Priority BCC states set of a desire is a subset of BCC states which has higher implication with respect to the particular desire.

Our definition of priority states set of a desire only consists of positive states since they provide a clear indication of the occurrence of activities. However, this definition can be altered if the negative states are also significant with respect to some desires. Analysts can choose this set for each desire using their background knowledge. Although the involvement of analysts is essential throughout this step, semi-automated tools can be developed to support them to make better decisions.
3.1.6 Generating sequence of situations

As given earlier, the definition of a situation at time t gives a 3-tuple \( \{d, A, E\} \). This section describes the method to generate a sequence of situations by using pre-processed BCC records and pre-processed ECC records to identify action set \( A \) and environmental context set \( E \) for each situation. Then the corresponding human desire \( d \) can be inferred.

Frequent patterns mining [25] is one of the common tasks in data mining, which aims at identifying a subsequence that appears in a data set with a frequency that is greater than or equals to some specified threshold. In our method, we define the frequency pattern as follows.

**Definition:** A frequency pattern \( (p) \) is a subset of positive BCC states that occurs frequently with the time. The pattern length is the number of BCC states in that pattern.

According to the definition, a frequency pattern only contains positive BCC states by eliminating both negative BCC states and ECC states. This will allow us to focus on activities been performed rather than activities not been performed. Also, it helps to reduce the pattern length since at a given instant there is fewer number of positive BCC states than the negative BCC states. However, this definition can be altered based on the significance of negative BCC states. Suppose that \( p_i \) represents the \( i^{th} \) pattern. Then, there exists a set of frequency patterns \( FP = \{p_1, p_2, ..., p_n\} \) in preprocessed BCC records where the number of patterns depends on the number of BCC and how related their positive states could be. It is possible that two frequency patterns share the same set of states and therefore, some longer patterns may include one or more shorter patterns.
As our first step of identifying actions set $A$, we obtain the set of frequency patterns $FP$ that appears in the pre-processed BCC records. We use the FP-Growth algorithm [25] for this task due to its high performance while any frequency pattern mining algorithm will be appropriate. Note that, this step can be fully automated. Once the set of frequency patterns $FP$ is obtained, the next step is to identify the original preprocessed BCC records where each of these patterns exists. Proposed algorithm to match the original preprocessed BCC records with the patterns in $FP$ is given in Algorithm 3.1. It takes the set of frequency patterns $FP$ and preprocessed BCC records $DS$ as the input and output labeled processed BCC record with frequency patterns as well as the set of frequency patterns used in labeling.

**Algorithm 3.1  Pattern selection algorithm for preprocessed BCC records**

| Data: $FP$: Set of frequency patterns, $DS$: Processed BCC records |
| Result: Processed BCC records labeled with patterns and final set of frequency patterns used. |

Read the Input datasets

| sort$_{FP}$ : Sorted $FP$ on descending order of pattern lengths |
| temp$_{DS}$ : $DS$ /* Initialize to original $DS$ */ |
| temp$_{FP}$ : sort$_{FP}$ /* Initialize to original sort $FP$ */ |

while temp$_{DS}$ is not empty do
  for each record $r$ in temp$_{DS}$ do
    for each pattern $pi$ in temp$_{FP}$ do
      if $r$ contains $pi$ then
        Label $r$ with $pi$
        Remove $r$ from temp$_{DS}$
      end
    end
  end

  temp$_{FP}$ : Get the new set of frequency patterns by taking remaining records in temp$_{DS}$ as the input

  sort$_{FP}$ : Set of (sort$_{FP}$ U temp$_{FP}$) sorted on descending order of pattern lengths

end

sort$_{FP}$ : Final set of frequency patterns used to label records in $DS$ sorted on descending order of pattern lengths
The labeled preprocessed BCC records for each frequency pattern can be further processed by combining the consecutive records with same pattern labels together to form a single record. We call the final set of labeled records as “concrete pattern records” and their time intervals as “concrete pattern intervals” since each of these labeled records corresponds to a particular frequency pattern. Note that the concrete pattern intervals are not overlapping. In the end, this will result in a sequence of concrete pattern records and the corresponding sequence of concrete pattern intervals. In the proposed method, we assume that the sequence of concrete pattern intervals introduces the time boundaries of situations, and the set of unique BCC states (both positive and negative) in concrete pattern records forms the set of actions occurred during that time interval. Moreover, we assume that the ECC states form the set of environmental contexts. Note that, the state of a particular ECC can be varied during a concrete pattern interval and therefore, each ECC state within a concrete pattern interval is recorded as the 3-tuple: \(<\text{minimum}, \text{maximum}, \text{mean}\>\).

At a given time \(t\), a person or group of people may have multiple desires. Therefore, the BCC states in a concrete pattern record may correspond to multiple desires. However, the positive BCC states in a concrete pattern record is the same as its labeled frequency pattern and the priority BCC states set of a desire only includes positive BCC states by definition. In other words, it is possible to introduce frequency patterns and desires mapping algorithm as given in Algorithm 3.2. The algorithm takes the final set of frequency patterns \(FP\) used to label preprocessed BCC records, the set of desires of interest \(DI\), the priority BCC states set of each desire \(DP\) and a user-defined threshold value \(T\) as the input and output the mapping between the frequency patterns and the desires.
The accuracy of this mapping can be improved by updating the priority BCC states set of existing desires or introducing new desires and corresponding priority BCC states sets by requirements analysts. The above process will generate a set of actions, set of environmental contexts and possible desires for each concrete pattern intervals.

![Algorithm 3.2](image)

According to the definition given in [10], a situation can contain at most one desire. Hence, two cases need to be considered when deriving sequences of situations.

- **Case 1:** If a concrete pattern interval mapped into only one desire, then that particular desire along with the set of actions and environmental contexts within the concrete pattern interval can be considered as one situation.
• **Case 2:** If a concrete pattern interval is mapped to multiple desires, it implies multiple situations had occurred simultaneously within the interval. Then, the set of actions during that time can be separated into subsets belonging to each desire based on the existence of them in the priority BCC states of desires. Note that, some actions can be common to multiple desires whereas some others do not belong to any desire. In the latter case, we can either ignore those actions or consider them as common to all the desires. A number of actions not belonging to any desire can be reduced by either updating the priority BCC states of desires or introducing new desires and corresponding priority BCC states sets. Note that the set of environmental contexts does not depend on desire. Each tuple of desire \( d \), a set of actions \( A \) corresponding to \( d \), set of environmental contexts \( E \) within the concrete pattern interval forms multiple simultaneous situations within the concrete pattern interval.

Note that the resulting sequences of situations from the above process may contain too many similar situations with same desire \( d \) and same or slightly different sets \( A \) and \( E \). Consideration of these similar situations as unique situations during the situation-transition structure deriving process will include redundant causal relationships and increase both time and space consumption. As a solution, we introduce a classification of situations into discrete groups. i.e., we further process the sequences of situations such that the situations with the same desire \( d \) and same or slightly different sets \( A \) and \( E \) are classified into one group. This will introduce discrete sets of situations and each set is given a unique name as \( S_{d1}, S_{d2} \ldots S_{dn} \). For a particular set \( S_{di} \), consider the common desire as \( d \), the union of actions sets belong to situations in \( S_{di} \) as \( A \) and the overall minimum, maximum, mean of environmental contexts
belong to situations in $S_{di}$ as $E$. Subsequently, the situations in the original sequence are renamed by their corresponding set names, which will result in a processed sequence of situations.

Figure 3.4  *Sample of preprocessed sequence of situations*

Figure 3.4 shows a sample of preprocessed sequence of situations. Such a sequence of situations can be used to define the situation-concurrency set as follows:

**Definition:** The situation-concurrency set of a situation sequence is the union of individual situations and groups of simultaneously occurred situations in that sequence of situations.

For example, the situation-concurrency set of the situation sequence given in Figure 3.4 includes three elements as \{S_A, S_C, S_D, S_F, \{S_A, S_C\}, \{S_B, S_F\}\}.

3.1.7 Deriving situation-transition structure

This section describes the process of identifying a causal relationship within the derived sequence of situations. Here we use the concept of Bayesian networks [26]; a well-known machine learning technique to infer the causal relationships between different factors based on observational data. Our methodology uses an algorithm called the Chow-Liu algorithm [27] as the baseline. In order to preserve information on the sequential ordering of
input data and to allow the presence of cycles, some modifications are made to the original algorithm.

Algorithm 3.3 describes the modified Chow-Liu algorithm. The algorithm takes a processed sequence of situations \textit{SITU\_SEQ}, the situation-concurrency set \textit{SITU\_CON} and a user-defined percentage of situation transitions to be selected $T$ as the input. The logical matrix is generated in order to calculate the mutual information between each pair of elements in the \textit{SITU\_CON}. The use of threshold value to decide the existence of edges allows the presence of cycles. Note that, the calculations of probabilities required a single \textit{SITU\_SEQ} travel which has linear time complexity.

A sample of the situation-transition structure is given in Figure 3.5. Note that this algorithm only considers the direct transitions from one situation to another in situation sequence when generating the transition structure. In other words, some it is possible to lose some information about the indirect transitions between situations in the sequence. As a solution, we have introduced a sliding window approach [28].

![Figure 3.5 Sample of situation-transition structure](image)
Data: SITU_SEQ: Sequence of situations with time, SITU_CON: Situation-concurrency set, PERCENTAGE: Percentage of situation transitions to be selected
Result: Situation transition structure

Read the Input datasets

num_Transitions: Number of situation transitions in SITU_SEQ. Initialize to zero.

SITU_AVAILABILITY = Mapping of each element in SITU_CON with an integer array of two elements. All elements initialize to zero.
SITU_PAIR_RELATION = Mapping of each possible pair of elements in SITU_CON with an integer array of four elements. All elements initialize to zero.

num_nodes = Cardinality of SITU_CON

MI = Two-dimensional floating-point array with size num_nodes × num_nodes to store Mutual Information
maxMI = Maximum value of Mutual Information
minMI = Minimum value of Mutual Information
T = Threshold value

/* Defining mappings */
for each x in SITU_CON do
  a = Integer array with two elements. All elements initialize to zero.
  /* a[0]=Number of occurrences where x does not appear in a pair of nodes in SITU_SEQ */
  /* a[1]=Number of occurrences where x appears in a pair of nodes in SITU_SEQ */
  Map x with a in SITU_AVAILABILITY
end

for each y in SITU_CON do
  b = Integer array with four elements. All elements initialize to zero.
  /* b[0]=Number of occurrences where x,y is not the current <parent,child> pair of situations in SITU_SEQ */
  /* b[1]=Number of occurrences where x is not the current parent but y is the current child situations in SITU_SEQ */
  /* b[2]=Number of occurrences where x is the current parent but y is not the current child situations in SITU_SEQ */
  /* b[3]=Number of occurrences where x,y is the current <parent,child> pair of situations in SITU_SEQ */
  Map <x,y> pair with b in SITU_PAIR_RELATION
end

/* (Continue on next page) */
current_parent = First situation in SITU_SEQ

for each current_child in SITU_SEQ starting from second situation do
  b = Integer array mapped to <current_parent, current_child> pair in SITU_PAIR_RELATION

  for each p in SITU_CON do
    if p ≠ current_parent and p ≠ current_child then
      otherPB = Integer array mapped to <p, current_child> pair in SITU_PAIR_RELATION
      otherPB[1] = otherPB[1] + 1
    end
  end

  for each c in SITU_CON do
    if c ≠ current_parent and c ≠ current_child then
      otherCB = Integer array mapped to <current_parent, c> pair in SITU_PAIR_RELATION
    end
  end

  num_Transitions = num_Transitions + 1

  /* Assign current child as the current parent for the next iteration */
  current_parent = current_child
end

Let a, b be an integer array.

for each x in SITU_CON do
  for each y in SITU_CON do
    if x ≠ y then
      b = Integer array mapped to <x, y> pair in SITU_PAIR_RELATION
      b[0] = num_Transitions - \sum_{i=1}^{3} b[i]
    end
  end

  a = Integer array mapped to x in SITU_AVAILABILITY
  a[0] = b[0] + b[1]
end

/* (Continue on next page) */
/* Generate situation dependency structure */

for each x in SITU_CON do
    Create a node in the graph.
end

for each x in SITU_CON do
    ax = Integer array mapped to x in SITU_AVAILABILITY
    x0 = ax[0]
    x1 = ax[1]

    for each y in SITU_CON do
        ay = Integer array mapped to y in SITU_AVAILABILITY
        y0 = ay[0]
        y1 = ay[1]

        if x ≠ y then
            b = Integer array mapped to <x,y> pair in SITU_PAIR_RELATION
            xy00 = \frac{b[0]}{\text{num}_\text{Transitions}}
            xy01 = \frac{b[1]}{\text{num}_\text{Transitions}}
            xy10 = \frac{b[2]}{\text{num}_\text{Transitions}}
            xy11 = \frac{b[3]}{\text{num}_\text{Transitions}}

            /* Calculate mutual information of <x,y> */
            if x0>0 and x1>0 and y0>0 and y1>0 and xy00>0 and xy01>0 and xy10>0 and xy11>0 then
                MI[x][y] = \sum_{i,j=0,1} xyij \times \log_2 \left( \frac{xyij}{xi \times yj} \right)
                Update maxMI and minMI
            end
        end
    end
end

T = (maxMI - minMI) \times \text{PERCENTAGE}
for each x in SITU_CON do
    for each y in SITU_CON do
        if MI[x][y] ≥ T then
            Create a directed edge from node x to y
        end
    end
end

Algorithm 3.3 Situation-transition structure generating algorithm
3.1.8 Mutual information values

Suppose X,Y represent any pair of nodes in the situation-transition structure. The mutual information of the directed edge between X,Y is,

\[ I(X,Y) = \sum_{x \in \text{values}(X)} \sum_{y \in \text{values}(Y)} \frac{P(x,y)\log_2 \frac{P(x,y)}{P(x)P(y)}} {P(x)P(y)} \]

where,

\(\text{values}(X) = \{\text{True}, \text{False}\}\)

**True**: When X is the parent node of the pair. i.e. Situation represented by X occurred just before other

**False**: When X is not the parent node of the pair. i.e. Situation represented by X does not occur just before other

\(\text{values}(Y) = \{\text{True}, \text{False}\}\)

**True**: When Y is the child node of the pair. i.e. Situation represented by Y occurred just after other

**False**: When Y is not the child node of the pair. i.e. Situation represented by Y does not occur just after other

\[ P(x,y) = \text{Joint probability of } X = x \text{ and } Y = y \]

\[ P(x) = \text{Probability of } X = x \ (0 < P(x) < 1) \]

\[ P(y) = \text{Probability of } Y = y \ (0 < P(y) < 1) \]

Note that the mutual information is directly proportional to the likelihood of this transition. That is, if the mutual information is high, the likelihood of the transition is also high and vice-versa.
CHAPTER 4. SITUATION-ORIENTED REQUIREMENTS ELICITATION

This chapter describes the proposed procedure to analyze the derived situation-transition structure and to elicit new requirements in terms of situations.

4.1 Analyzing situation-transition structure

4.1.1 Rooted situation-transition structure

The procedure of eliciting new requirements from the derived situation-transition structure starts with rearranging the structure into a rooted structure by selecting appropriate node as the root. Selection of root depends on the domain and set of available situations. We assume that the requirements analyst manually perform this task. Usually, the node representing the initial situation of the sequence of situations or the situation where all the actions and environmental context are in their default values is selected as the root. In cases where all the possible paths in the situation-transition structure cannot be traversed by starting from a single root node, it is required to select multiple root nodes. Once the root nodes are selected, all the possible paths starting from the root are listed. Figure 4.1 shows the set of possible paths in the situation-transition structure given in Figure 3.5. Here $S_A$ and $S_D$ are selected as the root nodes.

4.1.2 Definitions of property extracted terms

The analysis of situation-transition structures is benefited from the Causal Logic Model for Requirements Engineering by Moffett et al. [29]. This model introduced five major causal expressions as direct cause, lead to, terminate, sustain and prevent that can be
used to describe and analyze complex systems. Our methodology redefined these causal expressions terms to analyze the existing causal relationships in the derived situation-transition structures as given in Table 4.1.

### Table 4.1  
**Definitions of property extracted terms**

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct cause</td>
<td>If Situation $S_A$ directly causes Situation $S_B$, then the end of $S_A$ coincides with the start of $S_B$ in majority of observations.</td>
</tr>
<tr>
<td>Lead to</td>
<td>If Situation $S_A$ leads to Situation $S_B$, then it is possible to find intervening direct causes to make up a causal chain from $S_A$ to $S_B$.</td>
</tr>
<tr>
<td>Terminate</td>
<td>If Situation $S_A$ terminates Situation $S_B$, then the end of $S_A$ coincides with the end of $S_B$ in majority of observations.</td>
</tr>
<tr>
<td>Sustain</td>
<td>If Situation $S_A$ sustains Situation $S_B$, then $S_A$ does not end during $S_B$.</td>
</tr>
<tr>
<td>Prevent</td>
<td>If Situation $S_A$ prevents Situation $S_B$, then $S_B$ does not start during $S_A$.</td>
</tr>
</tbody>
</table>
4.2 New requirements elicitation using situation-transition structure

As mentioned earlier, the new requirements are elicited by analyzing the properties extracted from the situation-transition structure. Note that the elicited requirements can be in two types based on their influence towards the end user's behavior patterns.

1. Requirements that encourage and support the current behavior of the end-users.

2. Requirements that discourage and alter the current behavior of the end-users.

Although the first type of requirements directly follows the situation-transition structure properties, the elicitation of the second type of requirements needs background knowledge. Moreover, the functionalities of the elicited requirements must align with the objectives of the software-to-be. In other words, the same situation-transition structure can be used to elicit the requirements of different software products.

4.2.1 Definitions of requirements construction terms

In order to formally represent the new requirements, we define a set of requirements construction terms based on causal expressions. The definitions of a few such terms are given in Table 4.2. Each such definition of a requirements construction term may lead to derivation of one or more software system requirements. Introduction of these terms was influenced by the Goal Oriented Requirements Engineering approach proposed by Lamsweerde [30] to identify functional and non-functional goals of a system. However, our proposed methodology does not intend to identify system goals.

4.3 Case Studies

We have applied our proposed requirements elicitation methodology for real-life data sets in different application areas. It is important to note that the proposed approach does not
include any assumption on the existent of a prototype or previous version of the system-to-be. Hence, in this section we discuss three case studies in order to demonstrate that the proposed approach can be applied in any domain regardless of the existent of the system.

### 4.3.1 Case Study 1: Smart home energy management system

Smart* dataset [31] contains sensor data that were collected in a two-story, 1700 square foot home with three full-time occupants during May 01, 2012 – July 31, 2012 time period (306772 records). The data collecting components include sensors to record motion, door state, switch state, temperature, heat index, humidity, wind-chill, wind-speed and rainfall. We manually identified twelve possible desires such as prepare meal, wash dishes, sleep, laundry, etc.

The main aim of this case study was to determine new requirements for smart home energy management software that connects all the air-condition units and heaters in the house and is capable of adjusting their temperature setting automatically. It is obvious that developing such an automated system in real life is challenging since some of the requirements highly depend on the residents’ lifestyle, e.g., the temperature levels for each time period of the day. Therefore, it is difficult to find a common requirements set for this

---

<table>
<thead>
<tr>
<th>Set of requirements construction terms</th>
<th>Situation SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Require[Situation SD] [If Situation SC; SD direct cause SC then] earlier</td>
<td>SD</td>
</tr>
<tr>
<td>Achieve[Situation SD] [If Situation SC; SC leads to SD then] sooner-or-later</td>
<td>SD</td>
</tr>
<tr>
<td>Maintain[Situation SD] [If Situation SC; SD sustains SC then] always</td>
<td>SD</td>
</tr>
<tr>
<td>Avoid[Situation SD] [If Situation SC; SC prevents SD then] always not</td>
<td>SD</td>
</tr>
<tr>
<td>Stop[Situation SD] [If Situation SC; SC terminates SD then] sooner not</td>
<td>SD</td>
</tr>
</tbody>
</table>
system that gives high customer satisfaction. The traditional requirements elicitation approaches will only be able to find the most fundamental system requirements but they will not cover the requirements to satisfy all individuals and emerging situations. Note that this was the first attempt to create such software and hence neither a prototype nor an old version of the software existed during the requirements elicitation process.

First, we derived the situation-transition structure using the observational data collected by selecting the smart home as the domain and the current residents as the end-users. Next, we used it to elicit set of new requirements using the procedure given above. Figure 4.2 shows a part of this situation-transition structure. Each edge is annotated with the mutual information value of the transition. Table 4.3 listed sample requirements derived using this part of the structure.

4.3.2 Case Study 2: Smart personal assistant

In this case study, we used another smart home data set collected by Emmanuel Munguia Tapia for the thesis “Activity recognition in the home setting using simple and ubiquitous sensors” at MIT [32]. The data is collected using 80-100 reed switch sensors installed in a single-person apartment collecting data about human activity during March 27, 2003 – April 11, 2003 time period.

Our aim in this case study is to find human-centered requirements to develop an intelligent personal assistant (similar to Amazon Eco/Alexa or Siri app) to assist activities of daily living and improve the quality of life of end-users with special needs such as people with dementia. Note that these end-users may have different limitations in performing daily activities which are not visible in regular end-user groups. Some of the possible challenges
that may exist during the Requirements Engineering stage of this software project can be listed as follows:

1. The end-users are unable to provide clear expectations of the system. Sometimes the caregivers may be taken as an information source. However, they might not be able to specify the individualized needs and priorities of the end-user.

2. The parameters of the requirements depend on the individual and may change over time even for the same individual. For example, the time takes to complete a particular activity.

3. It is important to balance the care and the dignity of people with special needs [33]. For example, a person with early stage of dementia may not require guidance to perform each and every activity. Identifying the situations that need proper guidance and the situations where individual does not need such care is important to balance the care and dignity. Such balance could improve the quality of life of both the individual and the caregiver [34]. Moreover, the researches show the evidence of how important it is to continue the individual's regular patterns of living even after a person is diagnosis with dementia [33][34].

Hence, the regular Requirements Engineering approaches may not effective in this situation.
Figure 4.2 Part of the situation-transition structure of Smart* dataset
### Table 4.3  Sample set of requirements elicited from situation-transition structure in Figure 4.2

<table>
<thead>
<tr>
<th>Related situation transitions</th>
<th>Causal expressions</th>
<th>New requirements construction definitions</th>
<th>Derived new system requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wash_dishes → Relax_watch_TV</td>
<td>Wash_dishes leads to Relax_watch_TV</td>
<td>Achieve[Relax_watch_TV][If Wash_dishes; Wash_dishes leads to Relax_watch_TV then] sooner-or-later Relax_watch_TV</td>
<td>(A) If residents spend reasonable amount of time washing dishes in kitchen then adjust temperature and light in living room to relax situation level.</td>
</tr>
<tr>
<td>Prepare_meal</td>
<td>Prepare_meal sustains Wash_dishes</td>
<td>Maintain[Wash_dishes][If Prepare_meal; Prepare_meal sustain Wash_dishes then] always Wash_dishes</td>
<td>(B) If residents in kitchen, then maintain the water temperature of the kitchen faucet to reasonable level.</td>
</tr>
<tr>
<td>Prepare_meal and Cleaning_house → Relax_watch_TV</td>
<td>Prepare_meal and Cleaning_house sustains Relax_watch_TV</td>
<td>Maintain[Relax_watch_TV][If Prepare_meal and Cleaning_house; Prepare_meal and Cleaning_house sustain Relax_watch_TV then] always Relax_watch_TV</td>
<td>(C) During the time residents in kitchen and cleaning house then maintain the temperature and light in living room to relax situation level.</td>
</tr>
<tr>
<td>Sleep (Guest_bedroom) → Relax_watch_TV</td>
<td>Sleep (Guest_bedroom) leads to Relax_watch_TV</td>
<td>Achieve[Sleep(Guest_bedroom)][If Sleep(Guest_bedroom); Sleep (Guest_bedroom) leads to Relax_watch_TV then] sooner-or-later Relax_watch_TV</td>
<td>(D) If a resident is sleeping a reasonable amount of time in the guest bedroom then adjust temperature and light in living room to relax situation level.</td>
</tr>
<tr>
<td>No significant transition from Read to Prepare_meal or Read to Relax_watch_TV</td>
<td>Read prevents Prepare_meal and Relax_watch_TV</td>
<td>Avoid[Prepare_meal][If Read; Read prevents Prepare_meal then] always not Prepare_meal Avoid[Relax_watch_TV][If Read; Read prevents Relax_watch_TV then] always not Relax_watch_TV</td>
<td>(E) If the resident is reading in the master bedroom then start energy saving mode in the kitchen and the living room.</td>
</tr>
</tbody>
</table>
The original dataset is labeled with twenty two unique activities out of thirty four target activities. In order to get more realistic situation-transition structure with more situations we pre-processed the original data set so that contains transitions of twenty nine unique activities. Moreover, since the dataset is already labeled, it was directly used as the sequence of situations instead of redoing the data preprocessing given in Chapter 3. In addition, we again assumed that the domain did not include the software or a prototype.

Similar to the first case study, we derived the situation-transition structure using the labeled dataset and used it to elicit set of new requirements. Figure 4.3 shows a part of this situation-transition structure and the Table 4.4 listed some sample requirements derived using this part of the structure.

### 4.3.3 Case Study 3: Cooperative research environment

Cooperative Research Environment (CoRE) is a website that aimed for sharing published and unpublished internal research papers, comments and ideas in a research environment (Figure 4.4). The existing version of CoRE website was developed based on set of requirements elicited by the requirements analysts through traditional brainstorming techniques and there was no involvement of end-users within the process. Our aim was to find a list of new requirements for the next version of this website.
Figure 4.3 Part of the situation-transition structure of MIT dataset
### Table 4.4  Sample set of requirements elicited from situation-transition structure in Figure 4.3

<table>
<thead>
<tr>
<th>Related situation transitions</th>
<th>Causal expressions</th>
<th>New requirements construction definitions</th>
<th>Derived new system requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Taking medicine → Going out for work</td>
<td>Taking medicine direct cause Going out for work</td>
<td>Require[Take medicine] [If Going out for work; Taking medicine direct cause Going out for work then] earlier Take medicine</td>
<td>(A) Remind the user to take medicine before leaving the house.</td>
</tr>
<tr>
<td>Coming from shopping → Putting away groceries</td>
<td>Coming from shopping direct cause Putting away groceries</td>
<td>Achieve[Putting away groceries] [If Coming from shopping; Coming from shopping direct cause Putting away groceries then] earlier Putting away groceries</td>
<td>(B) Remind the user to put away groceries after coming from shopping.</td>
</tr>
<tr>
<td>Preparing lunch → Doing laundry</td>
<td>Preparing lunch sustains Doing laundry</td>
<td>Maintain [Preparing lunch] [If Doing laundry; Preparing lunch sustains Doing laundry then] always Preparing lunch</td>
<td>(C) Retain the kitchen working environment while user is doing the laundry.</td>
</tr>
<tr>
<td>Going to work → Taking medicine</td>
<td>Going to work terminates Taking medicine</td>
<td>Stop [Taking medicine] [If Going to work; Going to work terminates Taking medicine then] no Taking medicine after Going to work</td>
<td>(D) Once the user left home for work, update the status taking medicine is over and discourage any further attempt of taking medicine in the rest of the day.</td>
</tr>
<tr>
<td>No significant transition from lawn work to going to entertainment</td>
<td>Lawn work prevents going to entertainment</td>
<td>Avoid[Going to entertainment] [If Lawn work; Lawn work prevent Going to entertainment then] always not Going to entertainment</td>
<td>(E) Remind the user to finish lawn work before going to entertainment.</td>
</tr>
</tbody>
</table>
We considered the current version of the website as the domain and the participants of the website as the end-users. Records were collected on 65 end-users actions on current CoRE website interface such as button and link clicks, menu option selections, current and next webpages during September 16, 2014 – October 21, 2014 time period (10062 records) [35]. In this dataset the actions of each end-user can be uniquely identified through their login information. Moreover, we identified eighteen end-user desires such as login, view user profile, upload a paper, download a paper, and submit a comment. Again, we derived the situation-transition structure from the observational data and used it to elicit set of new requirements. Figure 4.5 shows a part of this situation-transition structure with mutual information value of the transition on each edge. Table 4.5 listed five sample requirements derived using this part of the structure. These five requirements were unknown when the exiting version of CoRE website was first developed.
Figure 4.5  Part of the situation-transition structure of CoRE dataset
Table 4.5  *Sample set of requirements elicited from situation-transition structure in Figure 4.5*

<table>
<thead>
<tr>
<th>Related situation transitions</th>
<th>Causal expressions</th>
<th>New requirements construction definitions</th>
<th>Derived new system requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Login → View_my_profile</td>
<td>Login leads to View_my_profile</td>
<td>Achieve[View_my_profile][If Login; Login leads to View_my_profile then] sooner-or-later View_my_profile</td>
<td>(A) Display the user profile on home page.</td>
</tr>
<tr>
<td>Upload_paper → View_my_profile → View_my_paper</td>
<td>Upload_paper leads to View_my_profile</td>
<td>Achieve[View_my_profile][If Upload_paper; Upload_paper leads to View_my_profile then] sooner-or-later View_my_profile</td>
<td>(B) Display the user profile and the list of papers uploaded by the user, once the user uploads a paper.</td>
</tr>
<tr>
<td>Download_paper → View_my_comment</td>
<td>Download_paper leads to View_my_comment</td>
<td>Achieve[View_my_comment][If Download_paper; Download_paper leads to View_my_comment then sooner-or-later View_my_comment</td>
<td>(C) Display a link to view submitted comments in download page.</td>
</tr>
<tr>
<td>Submit_comment → Logout</td>
<td>Submit_comment leads to Logout</td>
<td>Achieve[Logout] [If Submit_comment; Submit_comment leads to Logout then] sooner-or-later Logout</td>
<td>(D) Display a link to logout, once the user submits a comment.</td>
</tr>
<tr>
<td>Submit_comment → View_my_comment → Logout</td>
<td>Submit_comment leads to View_my_comment View_my_comment leads to Logout</td>
<td>Achieve[View_my_comment][If Submit_comment; Submit_comment leads to View_my_comment then] sooner-or-later View_my_comment Achieve[Logout][If View_my_comment; View_my_comment leads to Logout then] sooner-or-later Logout</td>
<td>(E) Display the submitted comments once the user submits a comment and then display the link to logout.</td>
</tr>
</tbody>
</table>
4.4 Discussion of proposed situation-oriented requirements elicitation approach

Some benefits of the proposed method over the existing methodologies have been observed as follows.

Requirements elicitation is based on situation-transition structure developed using collected data that targets specific end-user group in a particular domain. Hence, we believe that those elicited requirements reflect the actual end-user needs than the general requirements gathered using traditional approaches, and may gain customer satisfaction. Moreover, the proposed methodology can be used to elicit individualized requirements of the system-to-be if the behavior of each end-user can be distinctly identified using the raw data [36].

Generated situation-transition structure can provide an overall view of a domain of the system-to-be. For requirements analysts such structure is illuminating for identifying non-obvious indirect or hidden constrains of the requirements. Most of the existing requirements elicitation approaches lack this useful feature. For example, the residents in a smart home may not be aware that changing temperature level in one place will unexpectedly change the temperature in another place and result in wasted energy. The generated situation-transition structure in the proposed method can be used to elicit appropriate requirements in such situations.

Situation-transition structures can be reused for new system development in the similar application domain or in a product-line setting in the future. For example, the situation-transition structure generated for energy management software can also be used for requirements elicitation for other software systems such as home security system and personalized daily activity organizer. Continuous recording of observational data will be
helpful for identifying changes in end-user lifestyle to evolve existing system at run time [10][11].

Computationally rich nature of the proposed method leads to semi-automation that can reduce the workload of requirements analysts. In our case study we implemented a set of programs to automate the steps such as pre-processing raw data, identifying and generating sequence of situations and deriving the situation-transition structure. Properties of proposed methodology such as consideration of domains with multiple end-users, realistic data collection procedure and the possibility of semi-automation may increase its ability to accommodate real-world situations under different domains and conditions.

Despite of these significant benefits, the proposed methodology is not without limitations and potential caveats.

As mentioned earlier, identification of precise boundaries of actual situations is challenging. Additionally, there is no guarantee that the machine learning techniques used in the proposed method always provide the best possible situation-transition structure. Improvements in user observation by providing additional data collecting components will provide more accurate and distinct indication of user actions resulting in a more reliable situation-transition structure.

Requirements elicited from the proposed method highly depend on observational data; however, there is no guarantee that the observational data reflect the regular lifestyle of the end-users. For example, if data had been collected during a period of time where the user is in temporarily unfavorable health conditions, the derived situation-transition structure may not reflect the regular user lifestyle. Moreover, collected data can depend on the collection
period such as most of the days are holidays or only during winter. Therefore, special care needs to be taken on selecting the data collection period.

The duration of the entire requirements elicitation process may increase extensively depending on the characteristics of domain including new data sources, missing data and error in data collection. For example, if new residents arrive, the process needs to be repeated from the beginning in order to provide appropriate functionalities for those residents.

Security and privacy issues related to both end-users and domain of interest may limit the applicability of the method. One basic assumption in our work is that the system-to-be will be “custom built” while the economy of deploying of such a development method is out of the scope of our research. The premise is that in certain domains customer-built individualized services are essential to satisfying individualized requirements such as aging in place, disability support, etc. Further, the proposed approach does not specifically address a goal model such as Tropos [37]. We mostly stay in the human dimension [10] and only concern human desire in deriving situation-transition. In our view, goal model to be constructed for the system-to-be must conform to the situation-transition and satisfy the corresponding desires when rendering services to specific individuals.

4.5 Literature on requirements elicitation

At present, software systems grow fast in both size and complexity, and the requirements elicitation process becomes more difficult than never before. The rest of this section will discuss some of the recent studies in requirements elicitation related to our methodology.
4.5.1 Observation approach in requirements elicitation

Stakeholder observation is an established technique in requirements elicitation [38]. Recent studies have shown the advantages of unobtrusive observation although it is still not popular due to difficulties in analyzing it. For example, Brill, et al. [39] proposed a structured, unobtrusive observation approach for requirements elicitation of a context adaptive system based on the deviation of observed anonymous users’ behavior from predefined set of expected behaviors. Our method uses unobtrusive and unstructured observation which is known as more flexible than other category of observation [24]. Further, as oppose to [39], our method does not require the analyst to specify the initial set of requirements. In addition, our method is expected to find more specific requirements of the system that vary from user to user instead of finding a set of generalized requirements that are common to most users.

4.5.2 Situation awareness in requirements elicitation

Endsley, et al. [40] defines the term “Situation Awareness” (SA) as “the perception of the elements in the environment within a volume of time and space, the comprehension of their meaning and the projection of their status in the near future”. Several studies have been conducted relating SA with requirements elicitation. For example, Endsley, et al. study [40] refers the “Situation Aware Requirements” as the requirements of a system designed to support situation awareness, which can be determined using the goal structure generated by Goal-Directed Task Analysis (GDTA) based on the information gathered through interviews. Souza et al. [41] used the term “Awareness Requirements” to represent the requirements that refer to other requirements or domain assumptions and their success or failure at runtime.
Awareness Requirements are derived using the basic system requirements and the goal model constructed using the traditional requirements elicitation approaches. Our methodology use SA as an information resource for requirements elicitation. Thus, some of the elicited requirements can be treated as SA requirements. However, none of the requirements gathered in our method are Awareness Requirements by the definition of [41].

4.5.3 Individualized requirements elicitation

Although the idea of individualized requirements or variability of requirements to suit individual users is relatively new in computer science, it has been accepted and widely studied in other areas, especially in Human-Computer Interaction (HCI). Many such approaches, however, combining the concepts found in HCI and Requirements Engineering are labor-intensive. Sutcliffe et al. [42] discussed a three-layer framework that progress from general user group requirements to more personal and contextual requirements, with respect to two dimensions, location and time. Loeffler et al. [43] discussed the motivation towards “intuitively usable software” and addressed the user requirements in terms of human mental models. Their IBIS method (German for design of intuitive use with Image Schemas) integrates “image schemas” which represent recurring structures within human cognitive process into a Requirements Engineering and user-centered interface design process.

4.5.4 Crowdsourcing and requirements elicitation

Crowdsourcing is a novel distributed problem solving approach where the problem is solved through the involvement of a large number of people [44]. It has high potential in supporting Requirements Engineering and especially in requirements elicitation phase.
Snijders, et al. [45] presented the Crowd-Centric Requirements Engineering (CCRE) method that guides software product companies in effectively applying crowdsourcing throughout Requirements Engineering process where users become primary contributors targeting at higher quality requirements and high user satisfaction. From one point of view, our methodology can be considered as an application of crowdsourcing as it can be simultaneously applied on multiple end-user groups in a semi-automatic fashion.

### 4.5.5 Contextual requirements

Contextual requirements are defined as “the requirements that are only valid in a specific context” [46] and the term “context” is mainly referred to environmental contexts. Some studies presumed that these requirements are appropriate for the analysis of socio-technical systems and the changes in their operating environments [47]. However, we believe that the contextual requirements have not adequately treated human desires in the human dimension which indicates the main difference between the contextual requirements and the requirements gathered in our method.

### 4.5.6 Activity theory and requirements elicitation

Activity Theory (AT) in the field of psychology provides a framework for defining human activity as a system of multiple elements and their mediating relations which can be used to gather useful information such as identifying distinct stakeholders of the activity from the members of the community. Georg, et al. [48] proposed a socio-technical requirements elicitation approach that adopted the AT to elicit non-obvious social requirements in socio-technical systems where the authors adapt, formalize and combine AT
with existing goal/scenario modeling techniques. Note that, the definition of AT is concentrated on interactive human activities within a community and cannot be used to define the actions of isolated individuals [49] which limits its applicability of requirements elicitation in systems operated by single users. In contrast, our situation-oriented requirements elicitation methodology can be used in both single-user and multi-user systems development.
CHAPTER 5. SITUATION-ORIENTED REQUIREMENTS RISK MANAGEMENT

This chapter describes the proposed risk management procedure that identifies, analyzes, plans and monitors the risks of the human-system interaction related to the requirements derived in the previous chapter.

5.1 Human-centered risk management

Risk management is one of the essential steps in software development. Performing risk management early in the Requirements Engineering is beneficial since the effects of the risks are most unlikely to propagate to the later stages of development thus the overall damage will be reduced [50][51]. However, existing requirements risk management approaches have several limitations. For examples,

1. The term “risk” in Requirements Engineering has different meanings in different software project types [52]. Most often it mainly focuses on business and regulations restrictions such as development budget, unavailability of suppliers, deadlines of completion and violation of government safety regulations. However, one of the major concerns in modern socio-technical systems is the risk associated with the system interaction with the end-users and the surrounding environment. For example, in safety-critical systems risk is more related to the human aspects that include risk associated with humans, life threatening damages and injuries, etc.

2. Most of the existing risk management approaches require complete, well-defined requirements specifications [53]. However, modern software development is leaning toward the iterative and incremental development strategies (Agile methods) [54] which typically takes an exclusive set of requirements and develops that part of the software and
then move to another set of requirements. Although these approaches are fast, risk management is not established well due to their lack of documentation. Hence, the benefits of iterative and incremental development models have not been successfully propagated to some branches of software development such as safety-critical system development [55]. Moreover, these risk management approaches present limited abilities in handling the risks in changing environments caused by iterative and incremental software development.

3. Handling risks in Requirements Engineering is often narrowed to the assessment of risks which only identify and analyze the potential risks [56]. However, we believe that it is important to suggest possible solutions and alternatives in terms of requirements along with the identified risks such that the effect of the risk is not propagating to the design and implementation phases. In addition, a proper risk management approach must be able to introduce new requirements aiming to monitor the presence of unresolved risks in order to mitigate future damage.

In this section, we are proposing a systematic approach to risk management in Requirements Engineering. Here the term “risk” mainly represents the risks associated with the human-system interaction. In addition, we adopt the concept of feature models from software product lines to effectively represent the possible system functionalities along with the constraints between functionalities.

5.2 Features and feature models

The concept of features is widely used in Software Engineering. For example, features play the primary role in Feature-Oriented Software Development (FOSD) where
large-scale software systems are designed, constructed and customized by decomposing them in terms of the features [57]. In FOSD, a given set of features can lead to generating different software systems which share common features and differ in other features resulting in a Software Product Line (SPL) [57][58][59]. Pohl et al. [58] define SPL as “set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way”. While integrating SPL concepts with iterative and incremental development is prevalent ongoing research in Software Engineering [60][61][62][63][64] we believe that the set of software versions released by the iterative and incremental development process can be considered as SPL and hence the feature model representation may allow us to extend the proposed risk management approach to the iterative and incremental development paradigm.

According to the literature, the definition of the term “feature” is highly problem-oriented ranging from abstract format to more technical format [57] [65]. In their paper, Classen et al [65] compared several definitions of the term “feature” and proposed the following definition based on Zava and Jackson’s framework for Requirements Engineering [66] which states a problem-level feature as a set of related requirements, specifications and domain assumptions.

**Definition:** A feature is a triplet, \( f = (R, W, S) \), where \( R \) represents the requirements the feature satisfies, \( W \) the assumptions the feature takes about its environment and \( S \) its specification. [65]
Our approach follows the definition of Classen et al [65] and defines the feature as $R$ represents a requirement in terms of situations as we elicited from the situation-transition structure, $W$ represents the human behavior and the environmental context and $S$ represents the expected system functionality in an abstract manner. This definition is much convenient in our approach since it allows the transferring the information stored in situations to a more system design friendly representation with minimum information loss.

In 1990, Kang et al. [67] introduce Feature-Oriented Domain Analysis (FODA) and feature models. The feature model is a model that defines features of SPL and their dependencies which can be graphically represented by the feature diagrams (and-or trees) and the cross-tree constraints. Figure 5.1 shows a sample feature model. The basic feature models notations compose of four types of relationships between a parent feature and its child features and two types of cross-tree constraints as given in Table 5.1. In our approach, we use these basic feature models notations to represent the parent-child feature relationships and cross-tree constraints.

![Sample of feature model](image)

Figure 5.1 Sample of feature model
### Table 5.1  Basic feature model notations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Mandatory Symbol" /></td>
<td>Mandatory</td>
<td>Sub-feature is required.</td>
</tr>
<tr>
<td><img src="image" alt="Optional Symbol" /></td>
<td>Optional</td>
<td>Sub-feature is optional.</td>
</tr>
<tr>
<td><img src="image" alt="Alternative Symbol" /></td>
<td>Alternative</td>
<td>One of the sub-features must be selected</td>
</tr>
<tr>
<td><img src="image" alt="Or Symbol" /></td>
<td>Or</td>
<td>At least one of the sub-features must be selected</td>
</tr>
<tr>
<td><img src="image" alt="Requires Symbol" /></td>
<td>Requires</td>
<td>A requires B - selection of A in a product implies the selection of B</td>
</tr>
<tr>
<td><img src="image" alt="Excludes Symbol" /></td>
<td>Excludes</td>
<td>A excludes B - A and B cannot be part of the same product</td>
</tr>
</tbody>
</table>

### 5.3 Situation-oriented human-centered requirements risk management

This section describes the proposed risk management approach in details. Figure 5.2 shows the basic steps involved in this approach.

#### 5.3.1 Deriving situation-transition structure from observational data

As given in Chapter 3 [28], the generation of situation-transition structure starts with identifying the aim, domain of interest and the potential end-users of the software-to-be. The real-time observational data is then collected using various observational methods. Next, the situation-transition structure is generated by applying machine learning techniques to the collected observational data.
5.3.2 Eliciting set of requirements of the software-to-be

The initial set of requirements of the software-to-be is elicited by analyzing the possible paths of the generated situation-transition Structure as given in Chapter 4 [68]. It is important to note that each constructed requirement contains exactly two situations: the situation involved with the condition and the situation involved with the requirement construction term.
5.3.3 Deriving the initial set of features of the software-to-be

Once the set of requirements are elicited in terms of situations, we derive the features as \((R, W, S)\) triplets. Again, we are assuming that the requirement \(R\) is represented in terms of situations and the requirements construction terms we defined earlier. In other words, \(R\) represents the human-centered requirement of the system. The assumptions \(W\) include both human behavior factors and uncontrollable environmental factors. Also, the assumptions include the expected interaction between the human and the system as a set of transitions. The specification \(S\) includes the system functional or non-functional requirement that satisfies the human-centered requirement \(R\).

5.3.4 Generating the feature model of the software-to-be

The overall feature model of the software-to-be is generated in two steps. Each feature derived in the previous step is first decomposed into lower-level fine-grained sub-features. The resulting set of feature models is then composed into a single feature model which represents the overall software-to-be.

5.3.4.1 Feature model decomposition

Our feature model decomposition is developed on the following definitions and assumptions.

1. Each feature is represented as \((R, W, S)\) triplet which represents the requirement \((R)\), the assumptions \((W)\) and the specification \((S)\).

2. Each requirement represented by \(R\) parameter of the feature only comprises two situations.
3. For each situation,
   
   (a) Set of actions can be expressed as a sequence of action transitions between human and the system such that both first and last actions of the sequence are human actions.
   
   (b) Within the sequence of action transitions, at least one human action is unique to the particular situation and provides a clear indication of the start and end of the situation to the system.
   
   (c) Sequence of action transitions remains the same even if it is involved with multiple requirements. In other words, the sequence of action transitions is independent of the requirement. Hence, it is possible to keep a repository of action transitions sequences mapped with their corresponding situations.
   
4. Some environmental context in each situation related to a requirement R can be controlled by the system.

Algorithm 5.1 describes the proposed feature model decomposition. This algorithm takes a feature \(INIT\_FEATURE = (R,W,S)\) and the situation-action transition repository \(SITU\_ACT\_BASE\) as the input and returns the corresponding feature model. This feature model has \(INIT\_FEATURE\) as the parent node and fine-grained sub-features of \(INIT\_FEATURE\) as the child nodes in the second level. These sub-features are evaluated to determine whether they are mandatory or optional. Next, the required constraints are defined between sub-features based on the type of the requirements construction term used in the \(R\) parameter of the parent feature.
**Data:** FEATURE: A feature as (R,W,S) triplet, SITU_ACT_BASE: Repository of situations’ action transition sequences

**Result:** Feature model of FEATURE where each lower level represents more fine-grained sub-features and the constraints between sub-features

situ_cause: Situation connected to the condition of the requirement R of FEATURE.

situ_effect: Situation connected to the construction term of the requirement R of FEATURE.

situ_cause_actions: Action transition sequence of situ_cause from SITU_ACT_BASE

situ_cause_units: Set of units in situ_cause actions as <human_action,system_action,human_action> using the algorithm defined in Algorithm 5.2.

situ_cause_start: The unit of situ_cause_units where system_action marks the start of situ_cause.

situ_cause_end: The unit of situ_cause_units where system_action marks the end of situ_cause.

situ_effect_actions: Action transition sequence of situ_effect from SITU_ACT_BASE

situ_effect_units: Set of units in situ_effect_actions as <human_action,system_action,human_action> using the algorithm defined in Algorithm 5.2.

situ_effect_start: The unit of situ_effect_units where system_action marks the start of situ_effect.

situ_effect_end: The unit of situ_effect_units where system_action marks the end of situ_effect.

situ_cause_subfeatures: Set of subfeatures of FEATURE derived from situ_cause_units

situ_effect_subfeatures: Set of subfeatures of FEATURE derived from situ_effect_units

connect_subfeature: Sub-feature which contains specification S of FEATURE

FEATURE_subfeatures: Set of subfeatures of FEATURE

/* (Continue on next page) */
Consider FEATURE(R,W,S) for each i in situ_cause_units do
   Ri = R
   Wi = Starting and ending human actions of i
   Si = System action of i
   Add (Ri,Wi,SI) to situ_cause_subfeatures
end

/* Create sub-features of situ_effect */
for each i in situ_effect_units do
   Ri = R
   Wi = Starting and ending human actions of i
   Si = System action of i
   Add (Ri,Wi,SI) to situ_effect_subfeatures
end

/* Create the connect_subfeature */
Consider FEATURE(R,W,S)
Rc = R
Sc = S

Let REQUIRE, ACHIEVE, MAINTAIN, AVOID, STOP be the five possible requirements construction terms of R.

Let NOT(x) = Negation of x, x UNTIL y = Continue x until y is reached.

if R is REQUIRE then
   Wc= Starting human action of situ_cause_start, starting human action of situ_effect_start
end

if R is ACHIEVE then
   Wc= Ending human action of situ_cause_end, starting human action of situ_effect_start
end

if R is MAINTAIN then
   Wc= Starting human action of situ_cause_start, NOT(Starting human actions of situ_effect_end) UNTIL starting human action of situ_cause_end
end

if R is STOP then
   Wc= Starting human action of situ_cause_end, Starting human action of situ_effect_end, NOT[Starting human action of situ_effect_start]
end

if R is AVOID then
   Wc= Starting human action of situ_cause_start, NOT[Starting human action of situ_effect_start] UNTIL Starting human actions of situ_cause_end
end

connect_subfeature = (Rc,Wc,Sc) /* (Continue on next page) */
/* Complete FEATURE_subfeatures */
for each c in situ_cause_subfeatures do
  Add c to the FEATURE_subfeatures
end

Add connect_subfeature to the FEATURE_subfeatures

for each e in situ_effect_subfeatures do
  Add e to the FEATURE_subfeatures
end

root node = Create a node to represent INIT_FEATURE

for each f in FEATURE_subfeatures do
  Create a node to represent f and connect it to the root_node
end

/* Creating the required constraints and selecting mandatory, optional sub-features */
Consider FEATURE(R,W,S)
Mark connect_subfeature as Mandatory

Define require constraints between nodes representing sub-features and select mandatory as follows.

Mandatory_subfeatures: Mandatory sub-features in FEATURE_subfeatures

if R is REQUIRE then
  connect_subfeature requires situ_cause_start
  connect_subfeature requires situ_effect_start
  Mark situ_cause_start, situ_effect_start as Mandatory
end

if R is ACHIEVE then
  connect_subfeature requires situ_cause_end
  connect_subfeature requires situ_effect_start
  Mark situ_cause_end, situ_effect_start as Mandatory
end

if R is MAINTAIN then
  connect_subfeature requires situ_cause_start
  connect_subfeature requires situ_cause_end
  connect_subfeature requires situ_effect_end
  Mark situ_cause_end, situ_effect_start, situ_effect_end as Mandatory
end

/* (Continue on next page) */
Algorithm 5.1  Feature model decomposition

if $R$ is STOP then
  connect_subfeature requires situ_cause_end
  connect_subfeature requires situ_effect_end
  Mark situ_cause_end, situ_effect_end as Mandatory
end

if $R$ is AVOID then
  connect_subfeature requires situ_cause_start
  connect_subfeature requires situ_cause_end
  connect_subfeature requires situ_effect_start
  Mark situ_cause_start, situ_cause_end, situ_effect_start as Mandatory
end

while FEATURE_subfeatures is not empty do
  $m = \text{Next sub-feature from FEATURE_subfeatures}$

  if $m$ is start node then
    Mandatory_subfeatures = Call Algorithm 5.3 by passing
    FEATURE_subfeatures, Mandatory_subfeatures, $m$, Next
  end

  if $m$ is end node then
    Mandatory_subfeatures = Call Algorithm 5.3 by passing
    FEATURE_subfeatures, Mandatory_subfeatures, $m$, Prev
  end

  if $m$ is neither start node nor end node then
    Mandatory_subfeatures = Call Algorithm 5.3 by passing
    FEATURE_subfeatures, Mandatory_subfeatures, $m$, Next
    Mandatory_subfeatures += Call Algorithm 5.3 by passing
    FEATURE_subfeatures, Mandatory_subfeatures, $m$, Prev
  end
end

/* Creating the required constraints and selecting mandatory, optional sub-features */

for each $f$ in FEATURE_subfeatures do
  $f$ alternatives = Ask what? and how? to identify the alternative methods to satisfy $f$.
  Map $f$ with $f$ alternatives.
end
Algorithm 5.2  Dividing action transition sequence of a situation into a set of

\[ \text{<human_action, system_action, human_action>} \text{ units} \]
Algorithm 5.3  Mark mandatory sub-features

Data: FEATURE_subfeatures: Set of all the sub-features, m: Current sub-feature, direction: Next or Prev
Result: Updated FEATURE_subfeatures

if direction is Next then
  m_next: Take next sub-feature to m in the FEATURE_subfeatures
  while m_next is an end node do
    answer = Is m_next is required to confirm m happened?(YES/NO)
    if answer = YES then
      m requires m_next
      if m_next is not already marked as Mandatory then
        if m is marked as Mandatory then
          Mark m_next as Mandatory
        else
          Mark m_next as Optional
        end
      end
    end
  end
  m_next: Take next node to m_next in the FEATURE_subfeatures
end

Remove m from FEATURE_subfeatures

if direction is Prev then
  m_prev: Take previous node to m in the FEATURE_subfeatures
  while m_prev is a start node do
    answer = Is m_prev is required to confirm m will happen?(YES/NO)
    if answer = YES then
      m requires m_prev
      if m_prev is not already marked as Mandatory then
        if m is marked as Mandatory then
          Mark m_prev as Mandatory
        else
          Mark m_prev as Optional
        end
      end
    end
  end
  m_prev: Take previous node to m_prev in the FEATURE_subfeatures
end

Remove m from FEATURE_subfeatures
Note that we only define the “require” constraints between the sub-features of a feature model during the feature decomposition stage. The cross-feature models constraints and “exclude” constraints are handled during the feature composition stage. Moreover, the alternative methods to satisfy a particular sub-feature are represented as the child nodes in the third level. Figure 5.3 shows an example of a generated feature model.

![Feature Model Diagram](image)

**Figure 5.3 Sample of derived feature model from Algorithm 5.1**

### 5.3.4.2 Feature models composition

The development of the overall feature model given in Algorithm 5.4 starts by creating a node (parent node or root node) to represent the overall software-to-be and its child nodes to represent all the main objectives of the system. The objectives are evaluated in relation to the software-to-be to decide whether they are mandatory or optional. Next, connect the root nodes of the feature models (initial features) generated during the decomposition stage to the objective based on their relativity. An objective may be linked with multiple feature models as well as one feature model can be connected with multiple
objectives. Hence, the same feature model can be mandatory for one objective but can be optional to another objective. The assessment of the relativity can be decided by the requirements analysts.

Once the overall feature model is created, the cross-tree constraints must be identified between the objectives, initial features as well as sub-features. Here, we are introducing a bottom-up procedure to define the cross-tree constraints which starts from the sub-feature level. Note that a particular sub-feature can appear multiple times in different branches of the overall feature model, and the constraints that affect a sub-feature at one instant may affect others instances as well. Defining the cross-tree constraints to the sub-feature level must be done by carefully analyzing the possible requirements and the conflicts. The cross-tree constraints are then propagated to the initial features. For example, if $F_1$, $F_2$ are two initial features and if most of the sub-features of $F_1$ conflict with the sub-features of the $F_2$, that confliction can be represented as an exclude constraint between $F_1$ and $F_2$. Similarly, the constraints can be propagated to the objectives level as well.

5.3.5 Risk management of elicited requirements

Similar to the features, the definition of the term “risk” is highly problem-oriented. According to Cailliau et.al [50], the “risk” is generally defined as “an uncertain factor whose occurrence may result in some loss of satisfaction of some corresponding objective. It has a likelihood of occurrence, and one or several undesirable consequences associated with it”. Since our proposed requirements risk management approach is mainly focused on the human-system interface, we would like to re-define the term risk as follows.
**Definition:** A risk is a deviation from expected system behavior, human behavior, and environmental context which could lead to undesirable consequences.

In general, risk management comprises of four steps as identifying, analyzing, planning and monitoring the potential risks. The risk management is initially focused on the sub-feature level of the feature model which can be propagated upstream, and lead to risks at the high levels which represent initial features and the objectives. The following subsections explain our approach to cover these steps.

---

**Data:**

- **OBJECTIVES:** Set of objectives of the software-to-be,
- **FEATURE_MODELS:** Set of feature models generated from feature decomposition

**Result:** Overall feature model of the software-to-be

/* Generating overall feature model and selecting Mandatory and Optional objectives */

```
root_node = Create a node to represent the overall software-to-be.

for each obj in OBJECTIVES do
    Create a node to represent obj and connect it to the root node
    answer = Is obj is required to the software-to-be? (YES/NO)
    if answer = YES then
        Mark obj as Mandatory
    else
        Mark obj as Optional
    end
end

/* Connecting feature models to objectives */

for each fm in FEATURE_MODELS do
    fm_root_node_feature(R,W,S) = Take feature represented in the root node of fm
    for each obj in OBJECTIVES do
        answer = Is the two situations and the situation transition involved in the requirement R of
        fm_root_node_feature relevant to obj? (YES/NO)
        if answer = YES then
            Connect fm to obj and decide whether it is Mandatory or Optional
        end
    end
```

/* (Continue on next page) */
Algorithm 5.4  Feature model composition
5.3.5.1 Risk identification

The proposed approach identifies the risks related to sub-features by analyzing the possible consequences with unsatisfied assumptions and constraints. Algorithm 5.5 shows a generic procedure to follow which can be extended to more problem-specific, domain-specific formats in order to improve efficiency. Note that this procedure will only identify the risks related to individual sub-features. However, it is possible that risk may relate to a combination of sub-features even though each individual sub-feature is not directly related to it. Complex approaches need to identify these types of risks which is beyond the scope of this dissertation.

Data: FEATURE MODEL: Overall feature model of the software-to-be
Result: Set of potential labeled risks associated with the FEATURE MODEL, Mapping between sub-features and risks

risks_set: Set of potential risks
for each fm in FEATURE_MODEL do
    sub_f = Set of sub-features in fm
    for each s(Rs, Ws, Ss) in sub_f do
        s_constraints = Set of constraints related to s
        s_dependents = Subfeature related to the constraints in s_constraints
        /* Identifying the risks related to unsatisfied assumptions - human related risks */
        for each assumption in Ws do
            answer_1 = What if the assumption does not satisfy but Ss is executed?
            Possible choices:
            a. Ss will give same or acceptable outcome.
            b. Ss or sub-features in s dependents will give different or unexpected outcome.
            if answer_1 = b then
                break
            end
        end
    /* (Continue on next page) */
Once the potential risks of the sub-features are identified, the requirements analysts can categorize them to groups based on the severity of their consequences. Moreover, each sub-feature is mapped with its potential risks. The set of risks associated with an initial feature of the software-to-be is the union of the sets of risks mapped with its sub-features. We assume that the set of risks of an initial feature represents the risks associated with the corresponding requirement of that feature. Algorithm 5.6 presents the bottom-up approach to identify the risks associated with the requirements.

```
/* Identifying the risks related to unsatisfied specification - system related risks */

answer_2 = What if the Ws assumptions are satisfied but Ss is not executed?

Possible choices:
a. Domain status remains safe and sub-features in s_dependents will still execute and give same or acceptable outcome.
b. Either domain status becomes unsafe or sub-features in s_dependents will give different/unexpected outcome or both.

if answer_1 = a and answer_2 = a then
  s does not link to a risk caused by assumption.
end

if answer_1 = b or answer_2 = b then
  s may linked to a risk. Analyze further and add the possible risks to risks_set. Map the possible risks to s.
end

for each r in risks_set do
  Analyze the severity of the consequences of r.
  Label r as either High-Risk, Moderate-Risk or Low-Risk
end

Algorithm 5.5  Risk identification and categorization at sub-feature level
```
Algorithm 5.6  

**Risk identification of requirements**

5.3.5.2 Risk assessment

The proposed approach qualitatively assesses the risk of a particular sub-feature based severity of the consequences of the associated risks. The risk assessment of features can be derived using the assessment of the mandatory sub-features which we assume is the same as the risk assessment of the requirement involved with that feature. Here we ignore the effects of optional sub-features in order to simplify the process. Algorithm 5.7 shows this qualitative risk assessment of the requirements.

5.3.5.3 Risk planning and monitoring

Once the risks related to the requirements are identified and analyzed, the next step is to predict the possible solutions to eliminate or minimize those risks. Moreover, methods can be introduced to monitor the effects of partially defined risks in order to improve awareness. Algorithm 5.8 shows a possible approach to plan and monitor the risk.

---

**Algorithm 5.6 Risk identification of requirements**

Data: INIT_REQUIREMENTS: Initial set of requirements, FEATURE_MODEL: Overall feature model of the software-to-be, SUB_FEATURE_RISK_MAP: Mapping between sub-features and related risks

Result: Mapping between requirements and risks

for each r in INIT_REQUIREMENTS do
  f = Feature (R,W,S) in FEATURE_MODEL such that R=r
  sub_f = Set of sub-features of f
  Let f_risks = Set of risk mapped with f
  f_risks = \bigcup_{s_i \in sub_f} s_i\_risks
    where si\_risks = Set of risks mapped with sub-feature si using SUB_FEATURE_RISK_MAP
  Map r with f\_risks
end
Data: INIT_REQUIREMENTS: Initial set of requirements, FEATURE_MODEL: Overall feature model of the software-to-be, SUB_FEATURE_RISK_MAP: Mapping between sub-features and related risks, RISK_CLASSES: Types of risks

Result: Labeled requirements based on risk types

for each r in INIT_REQUIREMENTS do
    f = Feature (R,W,S) in FEATURE_MODEL such that R=r
    sub_f = Set of sub-features of f
    sub_f_classes: Set of risk types of the sub-features of f

    for each si in sub_f do
        si_risks = Set of risks mapped with si using SUB_FEATURE_RISK_MAP
        Let si_risks_classes = Set of risk types of si_risks

        for each r in si_risks do
            Get the type of r from RISK_CLASSES and add it to si_risks_classes
        end

        if si_risks_classes contains at least one High-Risk risk then
            si_class: High-Risk
        else
            if si_risks classes contains at least one Moderate-Risk risk then
                si_class: Moderate-Risk
            else
                if si_risks classes contains at least one Low-Risk risk then
                    si_class: Low-Risk
                else
                    si_class: No-Risk
                end
            end
        end

        Add si_class to sub_f_classes
    end

    if sub_f_classes contains at least one High-Risk sub-feature then
        f_class: High-Risk
    else
        if sub_f_classes contains at least one Moderate-Risk sub-feature then
            f_class: Moderate-Risk
        else
            if sub_f_classes contains at least one Low-Risk sub-feature then
                f_class: Low-Risk
            else
                f_class: No-Risk
            end
        end
    end

Map r with f_class
end

Algorithm 5.7 Qualitative risk assessment of requirements
Algorithm 5.8 Risk planning and monitoring of requirements

Data: LABELED_INIT_REQUIREMENTS: Labeled initial set of requirements from Algorithm 5.7, FEATURE_MODEL: Overall feature model of the software-to-be, SUB_FEATURE_RISK_MAP: Mapping between sub-features and related risks, RISK_CLASSES: Types of risks

Result: Updated labeled set of requirements

for each r in LABELED_INIT_REQUIREMENTS do
  if r labeled as No-Risk then
    Ignore r
  else
    while r is labeled as No-Risk or Acceptable do
      risky_sub_features = Select the sub-features corresponds to the label of r
      for each sf in risky_sub_features do
        Check the alternatives of sf in the lower level.
        if an alternative eliminate or minimize the involved risk then
          Update sf by adding that alternative directly to sf. Delete remaining alternatives.
        end
        if none of the alternatives eliminate or minimize involved risk or no alternatives listed then
          Update FEATURE_MODEL by introducing new sub-features and constraints to eliminate or minimize the involved risk.
          if sf is Low-risk then
            Update FEATURE_MODEL by introducing new sub-features to monitor the risk.
          end
        end
        if the risk involved with sf cannot be eliminate or minimize then
          Update FEATURE_MODEL by introducing acceptable sub-features to replace sf.
        end
      end
    end
  end
end

Update the SUB_FEATURE_RISK_MAP.
Update the label of r to a lower level or mark it as Acceptable.
This approach can also be extended to problem-centric and domain-centric approaches similar to the previous approaches. Note that the risk planning and monitoring can be done in parallel with the risk assessment stage.

5.4 Case Study

We conducted the requirements risk management case study on the five requirements elicited in the section 4.3.2. For each requirement, first, a feature model was created. Next, the five feature models were composed into a single model and the risk management steps were performed. Figure 5.4 to Figure 5.8 represent feature models and the following tables provide the detailed descriptions on the sub-features and their alternative. Figure 5.9 shows the composite model. This composite model only shows the sub-features involved with cross-tree constraints. Table 5.11 gives the identified risks, their severity labels and the related sub-features. Table 5.12 gives the qualitative risk assessment of the five requirements of the case study given in the section 4.3.2.

In order to mitigate the risks associated with the requirements, the following possible modifications were suggested.

1. The “take medicine” reminders must only be sent based on the medical dispenser status, and the system should continue sending after a time period if the status is not updated.
2. Use more observational data to distinguish the difference between going to work and going to entertainment. A direct verbal response from the user is one possible option.
3. Maintain the security feature without considering the user presence.
4. Repeat the refrigerator status update twice a day. Remind the user.
5. Introduce the energy saving mode when the user is present.
Feature $f_{\text{REQ}}(R,W,S)$

R = Require[Take medicine] [If Going out for work; Take medicine direct cause Going out for work then] earlier Take medicine.
W = User is able to perform basic activities such as moving, open cupboards, open doors etc. User memorizes the place of medicine container and the dose. Sensors are located to capture human actions. Sensors are located to capture the medicine dispenser status and other environmental context. All sensors are active and provide signals in real-time.
S = Remind the user to take medicine before leaving the house.

Figure 5.4 Case study 4.3.2 Require requirement feature model
<table>
<thead>
<tr>
<th>Sub-feature ID</th>
<th>Sub-feature</th>
<th>Alternatives</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>REQ_SF1</strong></td>
<td>R = Require[Take medicine] [If Going out for work; ...&lt;br&gt;W = User takes the office bag, laptop, phone, jacket, car keys ... etc. User moves to front door.&lt;br&gt;S = Mark ready to go for work.</td>
<td>REQ_SF1_ALT1 = List greeting, weather information, and appointments reminders. Read time of the day, item sensors, motion sensors</td>
</tr>
<tr>
<td><strong>REQ_SF2</strong></td>
<td>R = Require[Take medicine] [If Going out for work; ...&lt;br&gt;W = User moves to front door. User opens the front door.&lt;br&gt;S = Mark leaving home.</td>
<td>REQ_SF2_ALT1 = Read motion sensors</td>
</tr>
<tr>
<td><strong>REQ_SF3</strong></td>
<td>R = Require[Take medicine] [If Going out for work; ...&lt;br&gt;W = User moves to front door. User opens the front door.&lt;br&gt;S = Mark leaving home.</td>
<td>REQ_SF3_ALT1 = Read door sensor</td>
</tr>
<tr>
<td><strong>REQ_SF4</strong></td>
<td>R = Require[Take medicine] [If Going out for work; ...&lt;br&gt;W = User moves outside. User closes the front door.&lt;br&gt;S = Mark leaving home.</td>
<td>REQ_SF4_ALT1 = Read sensors outside porch</td>
</tr>
<tr>
<td><strong>REQ_SF5</strong></td>
<td>R = Require[Take medicine] [If Going out for work; ...&lt;br&gt;W = User moves outside. User closes the front door.&lt;br&gt;S = Mark leaving home.</td>
<td>REQ_SF5_ALT1 = Read door sensors and adjust heat, secure doors.</td>
</tr>
<tr>
<td><strong>REQ_SF6</strong></td>
<td>R = Require[Take medicine] [If Going out for work; ...&lt;br&gt;W = User takes the office bag, laptop, phone, jacket, car keys ... etc. User opens the dispenser and takes the medicine.&lt;br&gt;S = Remind the user to take the medicine if not taken for today.</td>
<td>REQ_SF6_ALT1 = Check the medicine dispenser status</td>
</tr>
<tr>
<td><strong>REQ_SF7</strong></td>
<td>R = Require[Take medicine] [If Going out for work; ...&lt;br&gt;W = User opens the bathroom medicine cabinet. User takes the medicine dispenser.&lt;br&gt;S = Mark at cabinet door open.</td>
<td>REQ_SF7_ALT1 = Read cabinet door sensors</td>
</tr>
<tr>
<td><strong>REQ_SF8</strong></td>
<td>R = Require[Take medicine] [If Going out for work; ...&lt;br&gt;W = User takes the medicine dispenser. User opens the dispenser.&lt;br&gt;S = Mark dispenser location states change.</td>
<td>REQ_SF8_ALT1 = Read item medicine dispenser</td>
</tr>
<tr>
<td>Sub-feature ID</td>
<td>Sub-feature</td>
<td>Alternatives</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
<td>--------------</td>
</tr>
</tbody>
</table>
| REQ_SF9       | R = Require[Take medicine] [If Going out for work; ...  
W = User opens the dispenser and takes the medicine. User closes the medicine dispenser  
S = Mark start taking medicine. | REQ_SF9_ALT1 = Read open medicine dispenser |
|               | R = Require[Take medicine] [If Going out for work; ...  
W = User closes the medicine dispenser. User keeps the dispenser in cabinet.  
S = Mark end taking medicine. Update current status | REQ_SF10_ALT1 = Read closed medicine dispenser |
| REQ_SF11      | R = Require[Take medicine] [If Going out for work; ...  
W = User keeps the dispenser in cabinet. User closes the cabinet door.  
S = Update of medicine dispense location. | REQ_SF11_ALT1 = Read presence of medicine dispenser |
| REQ_SF12      | R = Require[Take medicine] [If Going out for work; ...  
W = User closes the cabinet door. User walks away.  
S = Mark at cabinet door close. | REQ_SF12_ALT1 = Read cabinet door sensors |
Figure 5.5  Case study 4.3.2 Achieve requirement feature model
<table>
<thead>
<tr>
<th>Sub-feature ID</th>
<th>Sub-feature</th>
<th>Alternatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACH_SF1</td>
<td>R = Achieve[Putting away groceries] [If Coming from shopping;... W = User opens the front door. User moves to kitchen. S = Mark user is back.</td>
<td>ACH_SF1_ALT1 = Read motion sensors</td>
</tr>
<tr>
<td>ACH_SF2</td>
<td>R = Achieve[Putting away groceries] [If Coming from shopping;... W = User moves to kitchen. User keeps the groceries on table. S = Mark user location as kitchen.</td>
<td>ACH_SF2_ALT1 = Read motion sensors</td>
</tr>
<tr>
<td>ACH_SF3</td>
<td>R = Achieve[Putting away groceries] [If Coming from shopping;... W = User keeps the groceries on table. User walks away. S = Mark user kept new items on table.</td>
<td>ACH_SF3_ALT1 = Read sensors</td>
</tr>
<tr>
<td>ACH_SF4</td>
<td>R = Achieve[Putting away groceries] [If Coming from shopping;... W = User moves to kitchen. User opens refrigerator or freezer. S = Remind user to put away groceries</td>
<td>ACH_SF4_ALT1 = Read sensors</td>
</tr>
<tr>
<td>ACH_SF5</td>
<td>R = Achieve[Putting away groceries] [If Coming from shopping;... W = User opens refrigerator or freezer. User put groceries in. S = Mark open refrigerator or freezer.</td>
<td>ACH_SF6_ALT1 = Read door sensors</td>
</tr>
<tr>
<td>ACH_SF6</td>
<td>R = Achieve[Putting away groceries] [If Coming from shopping;... W = User put groceries in. User closes refrigerator or freezer S = Mark refrigerator or freezer filled.</td>
<td>ACH_SF7_ALT1 = Read item in refrigerator or freezer</td>
</tr>
<tr>
<td>ACH_SF7</td>
<td>R = Achieve[Putting away groceries] [If Coming from shopping;... W = User closes refrigerator or freezer. Users walks away. S = Mark refrigerator or freezer close.</td>
<td>ACH_SF7_ALT1 = Read door sensors</td>
</tr>
</tbody>
</table>
Figure 5.6  Case study 4.3.2 Maintain requirement feature model
Table 5.5  Case study 4.3.2 Maintain requirement feature model details

<table>
<thead>
<tr>
<th>Sub-feature ID</th>
<th>Sub-feature</th>
<th>Alternatives</th>
</tr>
</thead>
</table>
| MAIN_SF1      | R = Maintain [Preparing lunch] [If Doing laundry; ...  
W = User moves to laundry room. User opens washer.  
S = Mark user location as laundry_room.  | MAIN_SF1_ALT1 = Read the motion sensors |
| MAIN_SF2      | R = Maintain [Preparing lunch] [If Doing laundry; ...  
W = User opens washer. User put the cloths and closes it.  
S = Update the washer status.  | MAIN_SF2_ALT1 = Read the washer status. |
| MAIN_SF3      | R = Maintain [Preparing lunch] [If Doing laundry; ...  
W = User put the cloths and closes it. Turn on the washer  
S = Update the washer status.  | MAIN_SF3_ALT1 = Read the washer status. |
| MAIN_SF4      | R = Maintain [Preparing lunch] [If Doing laundry; ...  
W = Turn on the washer. User waits until washer finish. User walks away.  
S = Mark doing laundry start  | MAIN_SF4_ALT1 = Read the washer status. |
| MAIN_SF5      | R = Maintain [Preparing lunch] [If Doing laundry; ...  
W = User waits until washer finish. User opens the washer.  
S = Mark continue doing laundry  | MAIN_SF5_ALT1 = Read the washer status. |
| MAIN_SF6      | R = Maintain [Preparing lunch] [If Doing laundry; ...  
W = User opens the washer. User takes the cloths out.  
S = Update the washer status.  | MAIN_SF6_ALT1 = Read the washer status. |
| MAIN_SF7      | R = Maintain [Preparing lunch] [If Doing laundry; ...  
W = User takes the cloths out. User put cloths in dryer  
S = Update the washer status.  | MAIN_SF7_ALT1 = Read the washer status. |
| MAIN_SF8      | R = Maintain [Preparing lunch] [If Doing laundry; ...  
W = User put the cloths in dryer. Turn on the dryer  
S = Update the dryer status.  | MAIN_SF8_ALT1 = Read the dryer status. |
### Table 5.6  Case study 4.3.2 Maintain requirement feature model details (cont.)

<table>
<thead>
<tr>
<th>Sub-feature ID</th>
<th>Sub-feature</th>
<th>Alternatives</th>
</tr>
</thead>
</table>
| MAIN_SF9       | R = Maintain [Preparing lunch] [If Doing laundry; ... ]  
W = Turn on the dryer. User waits until dryer finish. User walks away.  
S = Mark drying start | MAIN_SF9_ALT1 = Read the dryer status. |
| MAIN_SF10      | R = Maintain [Preparing lunch] [If Doing laundry; ... ]  
W = User waits until dryer finish. User opens the dryer.  
S = Mark continue drying | MAIN_SF10_ALT1 = Read the dryer status. |
| MAIN_SF11      | R = Maintain [Preparing lunch] [If Doing laundry; ... ]  
W = User opens the dryer. User takes the cloths out.  
S = Update the dryer status. | MAIN_SF11_ALT1 = Read the dryer status. |
| MAIN_SF12      | R = Maintain [Preparing lunch] [If Doing laundry; ... ]  
W = User takes the cloths out. User walks away.  
S = Mark end doing laundry. | MAIN_SF12_ALT1 = Read the washer status. |
| MAIN_SF13      | R = Maintain [Preparing lunch] [If Doing laundry; ... ]  
W = Turn on the washer, NOT(User walks way for long period of time) UNTIL User takes the cloths out from dryer.  
S = Retain the kitchen working environment while user is doing the laundry. | |
| MAIN_SF14      | R = Maintain [Preparing lunch] [If Doing laundry; ... ]  
W = User moves to kitchen. User starts using kitchen appliances, items, drawers, etc.  
S = Mark user location as kitchen. | MAIN_SF14_ALT1 = Read the motion sensors |
| MAIN_SF15      | R = Maintain [Preparing lunch] [If Doing laundry; ... ]  
W = User starts using kitchen appliances, items, drawers, etc. User walks way for long period of time.  
S = Mark user start preparing lunch. | MAIN_SF15_ALT1 = Read sensors, time of the day |
| MAIN_SF16      | R = Maintain [Preparing lunch] [If Doing laundry; ... ]  
W = User walks way for long period of time. User not present in kitchen for considerable amount of time.  
S = Mark user end preparing lunch. | MAIN_SF16_ALT1 = Read sensors |
Feature $f_{STP} (R,W,S)$

R = Stop [Taking medicine] [If Going to work; Going to work terminates Taking medicine then] no Taking medicine after Going to work

W = User is able to perform basic activities such as moving, open cupboards, open doors etc. User memorizes the place of food containers. Sensors are located to capture human actions. Sensors are located to capture the refrigerator status and items status and other environmental context. All sensors are active and provide signals in real-time.

S = Once the user left home for work, update the status taking medicine is over and discourage any further attempt of taking medicine in the rest of the day.

Figure 5.7 Case study 4.3.2 Stop requirement feature model
Table 5.7  Case study 4.3.2 Stop requirement feature model details

<table>
<thead>
<tr>
<th>Sub-feature ID</th>
<th>Sub-feature</th>
<th>Alternatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>STP_SF1</td>
<td>R = Stop [Taking medicine] [If Going to work; ... W = User takes the office bag, laptop, phone, jacket, car keys ... etc. User moves to front door. S = Mark ready to go for work.</td>
<td>STP_SF1_ALT1 = List greeting, weather information, and appointments reminders. Read time of the day, item sensors, motion sensors</td>
</tr>
<tr>
<td>STP_SF2</td>
<td>R = Stop [Taking medicine] [If Going to work; ... W = User moves to front door. User opens the front door. S = Mark leaving home.</td>
<td>STP_SF2_ALT1 = Read motion sensors</td>
</tr>
<tr>
<td>STP_SF3</td>
<td>R = Stop [Taking medicine] [If Going to work; ... W = User moves to front door. User opens the front door. S = Mark leaving home.</td>
<td>STP_SF3_ALT1 = Read door sensor</td>
</tr>
<tr>
<td>STP_SF4</td>
<td>R = Stop [Taking medicine] [If Going to work; ... W = User moves outside. User closes the front door. S = Mark leaving home.</td>
<td>STP_SF4_ALT1 = Read sensors outside porch</td>
</tr>
<tr>
<td>STP_SF5</td>
<td>R = Stop [Taking medicine] [If Going to work; ... W = User closes the front door. User walks away. S = Mark left home.</td>
<td>STP_SF5_ALT1 = Read door sensors and Adjust heat, secure doors.</td>
</tr>
<tr>
<td>STP_SF6</td>
<td>R = Stop[Taking medicine][If Going to work; ... W = User closes the front door. User closes the medicine dispenser, NOT(User opens the dispenser and takes the medicine) S = Update the status taking medicine is over and discourage any further attempt of taking medicine in the rest of the day.</td>
<td>STP_SF6_ALT1 = Check the medicine dispenser status</td>
</tr>
<tr>
<td>STP_SF7</td>
<td>R = Stop[Taking medicine][If Going to work; ... W = User opens the bathroom medicine cabinet. User takes the medicine dispenser. S = Mark cabinet door open.</td>
<td>STP_SF7_ALT1 = Read cabinet door sensors</td>
</tr>
<tr>
<td>STP_SF8</td>
<td>R = Stop[Taking medicine][If Going to work; ... W = User takes the medicine dispenser. User opens the dispenser. S = Mark dispenser location states change.</td>
<td>STP_SF8_ALT1 = Read item medicine dispenser</td>
</tr>
</tbody>
</table>
Table 5.8  Case study 4.3.2 Stop requirement feature model details (cont.)

<table>
<thead>
<tr>
<th>Sub-feature ID</th>
<th>Sub-feature</th>
<th>Alternatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>STP_SF9</td>
<td>R = Stop[Taking medicine][If Going to work; ...</td>
<td>STP_SF9_ALT1 = Read open medicine dispenser</td>
</tr>
<tr>
<td></td>
<td>W = User opens the dispenser and takes the medicine. User closes the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>medicine dispenser</td>
<td></td>
</tr>
<tr>
<td></td>
<td>S = Mark start taking medicine.</td>
<td></td>
</tr>
<tr>
<td>STP_SF10</td>
<td>R = Stop[Taking medicine][If Going to work; ...</td>
<td>STP_SF10_ALT1 = Read closed medicine dispenser</td>
</tr>
<tr>
<td></td>
<td>W = User closes the medicine dispenser. User keeps the dispenser in cabinet.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>S = mark end taking medicine. update current status and</td>
<td></td>
</tr>
<tr>
<td>STP_SF11</td>
<td>R = Stop[Taking medicine][If Going to work; ...</td>
<td>STP_SF11_ALT1 = Read presence of medicine dispenser</td>
</tr>
<tr>
<td></td>
<td>W = User keeps the dispenser in cabinet. User closes the cabinet door.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>S = Update of medicine dispense location.</td>
<td></td>
</tr>
<tr>
<td>STP_SF12</td>
<td>R = Stop[Taking medicine][If Going to work; ...</td>
<td>STP_SF12_ALT1 = Read cabinet door sensors</td>
</tr>
<tr>
<td></td>
<td>W = User closes the cabinet door. User walks away.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>S = Mark at cabinet door close.</td>
<td></td>
</tr>
</tbody>
</table>
Feature $f_{AVD}(R,W,S)$

R = Avoid[Going to entertainment] [If Lawn work; Lawn work prevent Going to entertainment then] always not Going to entertainment
W = User is able to perform basic activities such as moving, open cupboards, open doors etc. User memorizes the place of food containers. Sensors are located to capture human actions. Sensors are located to capture the refrigerator status and items status and other environmental context. All sensors are active and provide signals in real-time.
S = Remind the user to finish lawn work before going to entertainment.

Figure 5.8  Case study 4.3.2 Avoid requirement feature model
Table 5.9  Case study 4.3.2 Avoid requirement feature model details

<table>
<thead>
<tr>
<th>Sub-feature ID</th>
<th>Sub-feature</th>
<th>Alternatives</th>
</tr>
</thead>
</table>
| AVD_SF1        | R = Avoid[Going to entertainment] [If Lawn work;...]  
W = User move to backyard. User takes the grass mower.  
S = Mark user is in backyard. | AVD_SF1_ALT1 = Read motion sensors outside backyard |
| AVD_SF2        | R = Avoid[Going to entertainment] [If Lawn work;...]  
W = User take the grass mower. User starts cutting grass.  
S = Mark grass mower absent. | AVD_SF2_ALT1 = Read grass mower sensor |
| AVD_SF3        | R = Avoid[Going to entertainment] [If Lawn work;...]  
W = User starts cutting grass. User stops cutting grass.  
S = Mark user start lawn work. | AVD_SF3_ALT1 = Read grass mower sensor |
| AVD_SF4        | R = Avoid[Going to entertainment] [If Lawn work;...]  
W = User stops cutting grass. User keeps the grass mower back.  
S = Mark user stop lawn work. | AVD_SF4_ALT1 = Read motion sensors, grass mower sensor |
| AVD_SF5        | R = Avoid[Going to entertainment] [If Lawn work;...]  
W = User keeps the grass mower back. User move inside.  
S = Mark grass mower present. | AVD_SF5_ALT1 = Read grass mower sensor |
| AVD_SF6        | R = Avoid[Going to entertainment] [If Lawn work;...]  
W = User starts cutting grass. User stops cutting grass. NOT[User takes the items phone, jacket, ... etc.] UNTIL [User stops cutting grass. User keeps the grass mower back.]  
S = Remind the user to finish lawn work before going to entertainment. |
| AVD_SF7        | R = Avoid[Going to entertainment] [If Lawn work;...]  
W = User takes the items phone, jacket, ... etc. User mover to front door.  
S = Mark ready to go for entertainment. | AVD_SF7_ALT1 = Read time of the day, item sensors, motion sensors |
| AVD_SF8        | R = Avoid[Going to entertainment] [If Lawn work;...]  
W = User moves to front door. User opens the front door.  
S = Mark leaving home. | AVD_SF8_ALT1 = Read motion sensors |
Table 5.10  Case study 4.3.2 Avoid requirement feature model details (cont.)

<table>
<thead>
<tr>
<th>Sub-feature ID</th>
<th>Sub-feature</th>
<th>Alternatives</th>
</tr>
</thead>
</table>
| AVD_SF9        | R = Avoid[Going to entertainment] [If Lawn work;...  
|                | W = User opens the front door. User moves outside. 
|                | S = Mark leaving home. | AVD_SF9_ALT1 = Read door sensors |
| AVD_SF10       | R = Avoid[Going to entertainment] [If Lawn work;...  
|                | W = User moves outside. User closes the front door. 
|                | S = Mark leaving home. | AVD_SF10_ALT1 = Read sensors outside porch |
| AVD_SF11       | R = Avoid[Going to entertainment] [If Lawn work;...  
|                | W = User closes the front door. User walks away.  
|                | S = Mark left home. | AVD_SF11_ALT1 = Read door sensors |
Figure 5.9  Smart personal assistant overall feature model
### Table 5.11  Smart personal assistant software risk identification

<table>
<thead>
<tr>
<th>Risk</th>
<th>Label</th>
<th>Related sub-features</th>
</tr>
</thead>
<tbody>
<tr>
<td>No reminder to take medicine.</td>
<td>High-Risk</td>
<td>REQ_SF1, REQ_SF6, REQ_SF9, REQ_SF10, STP_SF1, STP_SF9, STP_SF10</td>
</tr>
<tr>
<td>Provide false reminder to take medicine.</td>
<td>High-Risk</td>
<td>REQ_SF1, REQ_SF6, REQ_SF9, REQ_SF10, STP_SF1, STP_SF9, STP_SF10</td>
</tr>
<tr>
<td>Provide false reminder to discourage taking medicine.</td>
<td>High-Risk</td>
<td>STP_SF6</td>
</tr>
<tr>
<td>False identification as going to entertainment.</td>
<td>High-Risk</td>
<td>REQ_SF1, STP_SF1</td>
</tr>
<tr>
<td>Provide false reminder to keep the dispenser right place.</td>
<td>Low-Risk</td>
<td>REQ_SF11, STP_SF11</td>
</tr>
<tr>
<td>Cabinet door keep open for long time period.</td>
<td>Low-Risk</td>
<td>REQ_SF7, REQ_SF12, STP_SF7, STP_SF12</td>
</tr>
<tr>
<td>Setting the home condition to user left home status.</td>
<td>Low-Risk</td>
<td>REQ_SF5, STP_SF5, AVD_SF5</td>
</tr>
<tr>
<td>Keep the home condition to user left home status.</td>
<td>Low-Risk</td>
<td>ACH_SF1</td>
</tr>
<tr>
<td>Waste energy.</td>
<td>Low-Risk</td>
<td>ACH_SF5, MAIN_SF16</td>
</tr>
<tr>
<td>Extra food.</td>
<td>Low-Risk</td>
<td>ACH_SF6</td>
</tr>
<tr>
<td>Keep the home condition to user left kitchen status.</td>
<td>Low-Risk</td>
<td>MAIN_SF4, MAIN_SF13, MAIN_SF15</td>
</tr>
<tr>
<td>Keep the home condition to user present at kitchen status.</td>
<td>Low-Risk</td>
<td>MAIN_SF12, MAIN_SF13</td>
</tr>
<tr>
<td>Dislocated grass mower.</td>
<td>Low-Risk</td>
<td>AVD_SF2, AVD_SF5</td>
</tr>
<tr>
<td>Provide false reminder not to go out while doing lawn work.</td>
<td>Low-Risk</td>
<td>AVD_SF4, AVD_SF6</td>
</tr>
<tr>
<td>No reminder to finish the lawn work.</td>
<td>Low-Risk</td>
<td>AVD_SF6</td>
</tr>
<tr>
<td>Food turns into bad condition.</td>
<td>Moderate-Risk</td>
<td>ACH_SF4, ACH_SF5</td>
</tr>
<tr>
<td>False identification as going to work.</td>
<td>Moderate-Risk</td>
<td>ACH_SF5</td>
</tr>
<tr>
<td>Mitigate security features to user is present status.</td>
<td>Moderate-Risk</td>
<td>REQ_SF5, STP_SF5, AVD_SF5, ACH_SF1, ACD_SF1</td>
</tr>
<tr>
<td>Out of food.</td>
<td>Moderate-Risk</td>
<td>ACH_SF6</td>
</tr>
<tr>
<td>Dislocated medicine dispenser.</td>
<td>Moderate-Risk</td>
<td>REQ_SF8, REQ_SF11, STP_SF8, STP_SF11</td>
</tr>
<tr>
<td>Keep the home condition to user is present status.</td>
<td>Moderate-Risk</td>
<td>REQ_SF5, REQ_SF1, STP_SF5, STP_SF1, AVD_SF5, AVD_SF1, ACH_SF1</td>
</tr>
</tbody>
</table>
### Table 5.12  Smart personal assistant requirements qualitative risk assessment

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Qualitative Risk Assessment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Require[Take medicine] [If Going out for work; Take medicine direct cause Going out for work then] earlier Take medicine</td>
<td>High-Risk</td>
</tr>
<tr>
<td>Achieve[Putting away groceries] [If Coming from shopping; Coming from shopping direct cause Putting away groceries then] earlier Putting away groceries</td>
<td>Moderate-Risk</td>
</tr>
<tr>
<td>Maintain [Preparing lunch] [If Doing laundry ; Preparing lunch sustains Doing laundry then] always Preparing lunch</td>
<td>Low-Risk</td>
</tr>
<tr>
<td>Stop [Taking medicine] [If Going to work; Going to work terminates Taking medicine then] no Taking medicine after Going to work</td>
<td>High-Risk</td>
</tr>
<tr>
<td>Avoid[Going to entertainment] [If Lawn work; Lawn work prevent Going to entertainment then] always not Going to entertainment</td>
<td>Moderate-Risk</td>
</tr>
</tbody>
</table>

### 5.5 Discussion of proposed situation-oriented human-centered requirements risk management approach

Some benefits of the proposed method over the existing methodologies have been observed as follows. The proposed requirements risk management approach narrows down the requirements analyst’s focus to a particular human-system actions transition in the domain. We believe this will be a useful resource to the requirements analysts to find the unforeseen risks associated with unique human behavior patterns and make better decisions when planning the new requirements or constraints to eliminate or minimize such risks. Since the generated feature model is providing more information about the human-system interaction, it is possible to use the generated feature model as a prototype to generate test cases to verify and validate each release of the system as well as the final product. In addition, the flexibility of the feature models allows them to evolve with time and could be used as a source for future updates.
Despite these significant benefits, the proposed methodology is not without limitations and potential caveats. The proposed method highly depends on observational data hence the risks linked with unobserved situations may increase the uncertainty of the evaluated requirements. Use of different observational methods for an extended period of time could minimize such unobserved situations. Moreover, the manual process of creating human-system action sequences for each situation is ineffective or sometimes impossible in large and complex domains. Use of domain knowledge-bases to predict the expected system behavior for a given human action would be a promising solution for this problem.

In addition, defining the possible cross-tree constraints between feature models of large software may become complex. For example, with the possibility of multiple existences of a particular sub-feature in different branches, it is more likely to present confictions between unrelated sub-features. Careful analysis is needed to avoid or minimize such confictions so that they are not propagated to the risk assessment stage. A systematic procedure to analyze the constraint confictions will be a possible future work of the current study.

Finally, it is important to consider the differences between the proposed feature model-based approach to the existing Feature-Driven Development (FDD) [69] based iterative and incremental software development process. Similar to the proposed approach FDD is a client-centered process meaning that the main focus of the developers is to provide the features more valuable to the client. However, our features are represented in a more detailed format including the information on the domain and the human behavior while in FDD the features are client-valued system functions. Moreover, our overall domain model is
generated as a result of feature decomposition and composition process rather than merging the sub-domain area models.

**5.6 Literature on requirements risk management**

Although risk assessment can be done in any stage of the software development, shifting risk assessment to the Requirements Engineering phase can give many advantages when considering mitigation of severity and likelihood of risks in the initial software design [50][51]. The existing approaches to the risk assessment in Requirements Engineering stand in a wide range from organizations' regulations to complex machine learning techniques. These include both quantitative and qualitative approaches. The rest of this section will discuss some studies related to risk assessment in Requirements Engineering.

**5.6.1 Common risk assessment tools and techniques**

Brief description for some of the most commonly used risk assessment tools and techniques in software development is given below.

**5.6.1.1 Preliminary hazard analysis**

Preliminary Hazard Analysis (PHA) [56] introduced by the US Army is one of the commonly used tools for the initial risk assessment during the early stages of software development [70]. The idea here is to identify the hazardous states of a system and its implications through brainstorming and experts' background knowledge and experience. Since a PHA is performed at the early stages, it typically requires additional follow-up analyses to obtain a reliable risk assessment.
5.6.1.2 Failure mode and effect analysis

Failure Mode and Effect Analysis (FMEA) [56] is a table based approach developed to predict system reliability. FMEA especially adapted to software is known as Software FMEA or SFMEA. In general, FMEA is systematically exploring the effects of possible failure modes on a system and its environment through the forward search method which goes in time from failure to effect(s). The primary goal is to establish the overall probability that the product will operate without failure for a specific length of time. FMEA gives both quantitative and qualitative assessment to risk.

5.6.1.3 Fault tree analysis

Fault Tree (FT) [56] is a systematic investigation and graphical representation (tree structure) of the way in which a system can fail. The aim of Fault Tree Analysis (FTA) is to analyze the cause(s) of hazards. The root node of the tree structure is representing a hypothesized hazard. Each incremental level decomposes the previous level so that it lists more basic events to cause the parent node. The Boolean logic such as AND and OR are used to describe the combinations of contributing faults. The decomposition ends with the desired basic event is reached. Similar to FMEA, FTA can be used for both quantitative and qualitative risk assessment. The FTA especially defined for software is called Software FTA (SFTA).

5.6.1.4 Hazards and operability analysis

Hazards and Operability Analysis (HAZOP) [56] is a common technique used to assess the hazards of a system as well as its operability problems by exploring the effects of
any deviations from the design or operating intentions. HAZOP is capable of finding hidden and more complex hazards including human errors.

5.6.2 Probabilistic risk assessment and human reliability analysis

Probabilistic Risk Assessment (PRA) is one of the techniques used to analyze very complex systems such as an airliner, a nuclear power plant, and military applications. PRA is used to predict the future risk from observation of past performance and/or through the statistical data about the individual sub-systems of the application although the final application does not exist yet [56]. However, the statistical background of PRA may introduce problems with more interactive applications where the impact of human operators upon system behaviors is inevitable [71]. In complex applications, it would be difficult to generalize the impact of an individual's cognitive and perceptual factors on error probability [71]. As a solution, Human Reliability Analysis (HRA) was proposed. HRA quantifies the probability of human errors through the techniques defined for PRA [56][71]. It can also be used to identify the possible alternative steps or activities in the process in order to reduce the probability of human error [56]. The early HRA approaches THERP [62] and HEART [63] ignore some significant factors such as the working environment, organizational context and individual cognitive abilities that directly impact human errors [61][64]. Some of the subsequent HRA techniques used Performance Shaping Factors (PSF) to estimate the human error probability (HEP) [71]. Here the term PSF implies various factors that affect human performance and could change the likelihood of a human error. The effects of PSF were determined with reference to underlying cognitive and perceptual models of performance such as ATHEANA (US NRC, 1996) [75] and CREAM [74]. In recent studies, the researchers have questioned the applicability of HRA in more complete interactive
applications. For example, in [71] Johnson describes the limitations of integrating PRA techniques in HRA related to four major factors: determinism, induction, deduction, and context.

Similar to PRA and HRA, our proposed risk management approach is primarily based on observation and can be used to predict the prospective risks. However, the situation-oriented feature of the proposed approach provides a theoretical basis and a practical tactic to embed the individual's cognitive and perceptual factors to risk assessment even in more complex interactive applications.

5.6.3 Risk matrix

Risk matrix maps out different levels of risk within the cells that denote particular combinations of likelihood and consequence. The impact of mitigations can then be shown as movement between the columns and rows of the matrix [76]. Although risk matrix is one of the earliest techniques in risk assessment, recent studies are still implementing them connected with modern techniques. For example, Ancel et al. [77] introduced a real-time risk assessment approach for Unmanned Aircraft System (UAS) Traffic Management using Bayesian Belief Networks (BBNs) [78]. UASs have been the focus of many commercial and civilian applications including infrastructure monitoring, delivery of goods, precision agriculture, public safety, search and rescue, disaster relief, weather monitoring, etc. However, the safety of the people on the ground, who are not directly involved with the aircraft operation, is a key aspect of such applications especially when operated in populated areas. The risk assessment proposed by this study provides risk metrics associated with casualties in real-time. These risk metrics were used to classify and assess the risk with
known likelihood and severity. The minimum acceptable level of risk was determined by severity categories low, medium and high.

5.6.4 Goal-oriented risk assessment approaches

While above risk assessment approaches can apply for any software developmental phase the Goal-Oriented Risk Assessment Approaches are specifically defined for Requirements Engineering phase. These approaches use the concepts and properties of Goal-Oriented Requirements Engineering (GORE) framework. In [50] Cailliau et.al presents a probabilistic framework for goal specification and obstacle assessment. Obstacle analysis is a goal-oriented form of risk analysis where an obstacle to a goal is a precondition for non-satisfaction of that goal. The quantitative risk assessment technique presented in [50] extends the GORE framework by introducing a probabilistic layer that allows behavioral goals to be characterized in terms of their estimated and required degrees of satisfaction. Asnar et.al [79] introduces a three-layered risk assessments model founded on three main concepts: asset, event, and treatment. Assets, modeled in terms of goals, are analyzed and related to external events that can influence negatively (i.e., risk) their satisfaction. Treatments are then introduced to mitigate the effects of such events. In [51] Tangsaksant and Prompoon present a framework for risk assessment during analysis and design phase which identifies risk from object behaviors and their interactions in order to satisfy the software functions that support the organizational goal. Here the software functional, structural and behavioral model is produced by analyzing organizational goal through GORE. Risk assessment is performed by considering the object behaviors and their interactions. The risks caused by every single object are then integrated into the scenario risk factor and a functional risk factor of the functional model. These factors can be used to manage and control risk into a fine-grained
level in order to support an organizational goal. Islam [80] proposed a Goal-Driven Software Development Risk Management Model (GSRM) which is a combination of four layers named goal layer, risk-obstacle layer, assessment layer, and treatment layer. The goal layer includes steps to identify, elaborate and model the goals based on the early development components from the perspective of project success. The risk-obstacle layer identifies the potential software development risk factors as obstacles that have a negative impact on the project's goals. The assessment layer analyzes the risk event caused by the identified risk factors where each risk event is characterized by its likelihood and severity. An obstruction link is established from risk event to the specific goal it obstructs. The risks are prioritized based on their likelihood, severity, and influence towards goal dissatisfaction through obstruction link. Finally, the treatment layer identifies the possible control actions and selects the most suitable ones to mitigate the risks in order to achieve the project goals.

We believe that situation-oriented approach to risk management adds dynamic to the risk assessment in a broad spectrum since situation is a descriptive unit that encapsulates human mental and behavioral status along with environmental properties in the domain.

5.6.5 Machine learning based risk assessments

In recent studies, researchers have focused on using various machine learning techniques to develop risk assessment approaches suitable for large, complex software development projects. Bayesian Belief Networks (BBNs) [78] is one of the most commonly used machine learning techniques for risk assessment due to its ability to handle uncertainty.

In [81] Kumar et al. proposed a probabilistic software risk estimation model based on BBNs that focuses on the top software risk indicators for risk assessment in software development projects. Once the top-ranked software risk indicator metrics for the project
have been selected, a BBN is constructed using the historical data, experimental observation, and domain experts' experience. The risk assessment is performed by using the constructed BBN. In [82] Fan et al. designed a procedure to incorporate BBNs in a continuous monitoring loop to support the decision making process of risk management. Here, BBNs is used to identify the source of the risks as well as to model the uncertainties by providing probabilistic estimations of risks. These estimations were recalculated and updated when new data is available in the monitoring loop. Lee et al. [83] discussed a scheme for large engineering project risk management using a BBN. Internal risks such as software design changes, design manpower, and raw material supply as well as external risks such as currency exchange rate were taken into account for both large-scale and medium-sized shipbuilding companies. Once the associated risks were identified and classified, the risk level of each risk was determined using the degree of loss and the probability of occurrence. The dataset was then modified using a given risk matrix in order to apply a BBN analysis. Next, the BBN was constructed by structural learning and the impact of each risk to the project performance was measured through a sensitivity analysis of the constructed BBN. Once the set of risks to be controlled were selected, the conditional probabilities of project performance risks were measured in relation to the selected risks change and were used as a feedback loop for the risk management. Hu et al. [84] introduced a framework for software project risk management using Bayesian Networks with Causality Constraints (BNCC). The main objective of the framework was to perform a causality analysis between risk factors and project outcomes to achieve more effective risk control. This study by Hu et al. [84] emphasizes the benefits of finding the causal relationships between risk factors and project outcomes in contrast to finding the correlation between them.
Similar to [83], we also introduced loops in situation-transition structure even though the original Bayesian networks are defined as acyclic.

5.6.6 Risk assessment in continuously improving and changing software development

In [55] Ge et al. discussed an iterative approach for safety-critical software development. Although the authors argued that the lightweight and iterative approach allied with Agile methods can improve the development of safety-critical systems, they believed that direct application of Agile methods in safety-critical systems development still requires more certification. In their work, they address the possibility of applying the up-front design in safety-critical software development and achieving safety objectives. Moreover, a methodology is presented to develop both the software system and a safety argument iteratively. Six-Sigma is one of the quality management approaches for systems with continuous improvements [56]. It measures the degree by which a process deviates from its goal [85]. [86] explains how Six-Sigma can be used for reliability and safety analysis of a system. In [87] Fu et al. proposed a Design Structure Matrix (DSM) based probabilistic model to evaluate the risk of change propagation from requirements to software architecture by estimating the schedule and cost of the development. The risk of change propagation for each component of the software is predicted using this model which enables module reorganization instead of redesigning.
CHAPTER 6. SITUATION-ORIENTED REQUIREMENTS EVALUATION AND PRIORITIZATION

Various requirements prioritization techniques have been introduced to support requirements evaluation and prioritization based on factors such as their importance for the overall system functionality, limits on budget, required time, potential risk, etc. [88]. This chapter describes our proposed approach to evaluate and prioritize requirements to balance the human and system viewpoints.

6.1 Human factors in requirements prioritization

As we mentioned earlier, each individual human being is unique and hence each individual end-user has a set of prioritized requirements imposed upon the particular software, and the user satisfaction depends on whether or not those requirements are fulfilled by the software. It is also important to note that the requirements that have a higher priority to one end-user can be entirely unimportant to another. Based on these observations, we believe that the requirements prioritization process can be made more effective by considering the information about the uniqueness of the individual’s prioritization of those requirements [9]. In this section, we propose a new approach to prioritizing requirements on the individual basis by considering the importance of the requirement to a particular end-user (encoded as the \textbf{Human Importance Factor}) using the information included in the situation-transition structure defined in Chapter 3. A requirement will be assigned higher priority if its Human Importance Factor is high [89].
According to the definition, the Human Importance Factor measures the importance of the requirement to the end-user. Here we assume that the importance or relevance of the requirement to the end-user is directly proportional to how often the user required the functionality provided by that requirement. As given in Chapter 4, the elicited requirements are related to one or more situation transitions in the situation-transition structure. Therefore, based on our previous assumption, it is possible to claim that the importance of a particular requirement is directly proportional to the likelihood that the user engaged in the situation transitions which leads to eliciting that particular requirement. As described in Chapter 3 the mutual information can be used as a quantity that represents the likelihood of this transition. Hence, we can use mutual information values of the transitions to evaluate the Human Importance Factor of a particular requirement.

**Definition:** Consider a requirement $R$. Let $p$ denote the path in the situation-transition structure that leads to eliciting $R$ and $I(X,Y)$ denote the mutual information value of a transition $<X,Y>$ in the path. Then, human importance factor of $R$, $HIMP_R$ can be defined as,

$$HIMP_R = \sum_{<X,Y> \in \text{Set of transitions associated with path } p \text{ that is related to } R} I(X,Y)$$

**6.2 System factors in requirements prioritization**

Most of the traditional requirements prioritization methods mainly focus on their importance for the overall system functionality [88]. In this section, we propose a new approach to prioritizing requirements by considering the importance of the requirement to a system (encoded as the System Importance Factor) using the information included in overall
system feature model defined in Chapter 5. A requirement will be assigned higher priority if its System Importance Factor is high.

According to the definition, the System Importance Factor measures the importance of the requirement to the overall system. Here we assume that the importance of a requirement to the system is directly proportional to how often the functionalities provided by that requirement can be reused when comprising the other requirements to the system. As given in Chapter 5, the elicited requirements can be represented as features and for each feature, a feature model can be derived to represent the sub-features and their alternatives. The same sub-feature can exist in different features and hence there are multiple existences of that sub-features in the overall system feature model. Therefore, based on our previous assumption, it is possible to claim that the importance of a particular requirement is directly proportional to a number of occurrence of the related sub-features. In other words, if there is a particular sub-feature with multiple existences in the feature model, the requirements related to that sub-feature have higher priorities.

**Definition:** Consider a requirement R. Let FR denote the feature model of R and \( FR_{SUB} \) denote the set of sub-features of FR. Then, system importance factor of R, \( SIMP_R \) can be defined as,

\[
SIMP_R = \sum_{s \in FR_{SUB}} \text{Number of occurrences of } s \text{ in overall feature model}
\]

### 6.3 Risk factor of requirements

In Chapter 5 we introduced a new situation-oriented risk management approach which mainly focuses on the risk associated with the human-system interactions. We believe
that the risk assessment (encoded as the **Risk Factor**) used in Chapter 5 can also be used to evaluate and prioritize the requirements. As Algorithm 5.7 is providing a qualitative risk assessment of requirements, we define the risk factor as a mapping between qualitative risk assessment and a numerical scale. A requirement will be assigned higher priority if its Risk Factor is low [89].

**Definition:** Consider a requirement $R$. Let $Q$ denote the qualitative risk assessment of $R$ by Algorithm 5.7. Then, the risk factor of $R$, $RISK_R$ can be defined as,

$$RISK_R = f(Q)$$ such that $f$: Qualitative Risk Assessment $\rightarrow$ Numerical Scale

### 6.4 Requirements prioritization

According to the proposed approach, the priority of a requirement $R$ is high when its Importance factors are high or the Risk factor is low. Hence, the priority of the requirement $R$ (encoded as the **Priority(R)**) can be estimated as follows:

$$\text{Priority}(R) = \begin{cases} 
    k_1HIMP_R + k_2SIMP_R & \text{If } RISK_R > 0 \\
    k_1HIMP_R + k_2SIMP_R & \text{Otherwise}
\end{cases}$$

where, $k_1, k_2$ are positive constants.

Requirement analyst can select positive constants $k_1, k_2$ such that the priorities values lie within a pre-defined range. Note that these constants can be used to bias the requirements prioritization towards the human or system perspectives. Similar to our proposed requirements elicitation and risk management approaches, it is important to note that the proposed requirements prioritization approach is not aiming to completely eliminate the use
of other approaches, but instead provide additional information to the requirements analysts to make better decisions during the requirements prioritizing phase.

6.5 Iterative and incremental software development

Iterative and incremental software development is a method of software development that is modeled around a gradual increase in function additions and a cyclical release and upgrades pattern [54] Figure 6.1. This approach is widely used for large projects. Iteration includes designing, implementing, testing, evaluating and releasing the software with a selected set of requirements. The number of iterations of the entire software development depends on the application, the requirements and the effectiveness of the earlier iterations.

Figure 6.1 Iterative and incremental software development process
6.5.1 Extension of proposed human-centered Requirements Engineering approaches to iterative and incremental software development

One main characteristic of the iterative and incremental software development is the recurrent involvement with the end-users. We believe that our proposed human-centered Requirements Engineering approaches can be extended to evolve with such iterative and incremental software by considering the released software version at the end of each developmental iteration as a part of the domain of the next iteration. In other words, the domain is updated by adding the software version. It is important to note that such change to the domain may change the human-system interaction. The human-system interactions in the updated domain may or may not follow as expected. It could be possible that these changes in the domain may lead to a new set of requirements as well as new risks and need to take into account when selecting the functionalities of the future developmental iterations.

It is possible to get a more reliable picture of the updated domain by performing the original form of the domain observation and deriving a new situation-transition structure to indicate the current human-system behavior. Any deviation from the previous iteration’s expected transitions of human-system actions indicates an unpredicted behavior. Hence the new set of requirements needs to be elicited and re-evaluated for risk management. If there is a possibility of risk in such a case, planning and monitoring can be repeated to eliminate or mitigate the damage. In exceptional cases, this might implies the need of removing already implemented functionalities which may cause a negative impact to the overall system development since the most of the functionalities related to such unforeseen behavior may link with other implemented functionalities of the system.
6.6 Case study

We conducted the requirements prioritization case study on the five requirements elicited in the section 4.3.2. Table 6.1 gives the requirements, the situation transitions involved in each requirement and the human importance factors.

Table 6.1  Smart personal assistant requirements human importance factor

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Related Situation-Transitions</th>
<th>Human Importance Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Require[Take medicine] [If Going out for work; Take medicine direct cause Going out for work then] earlier Take medicine</td>
<td>Take medicine → Going out for work</td>
<td>0.0291359538</td>
</tr>
<tr>
<td>Achieve[Putting away groceries] [If Coming from shopping; Coming from shopping direct cause Putting away groceries then] earlier Putting away groceries</td>
<td>Coming from shopping → Putting away groceries</td>
<td>0.0119613639</td>
</tr>
<tr>
<td>Maintain [Preparing lunch] [If Doing laundry; Preparing lunch sustains Doing laundry then] always Preparing lunch</td>
<td>Preparing lunch → Doing laundry → Putting away laundry → Preparing lunch</td>
<td>0.0156759941</td>
</tr>
<tr>
<td>Stop [Taking medicine] [If Going to work; Going to work terminates Taking medicine then] no Taking medicine after Going to work</td>
<td>Taking medicine → Going to work</td>
<td>0.0291359538</td>
</tr>
<tr>
<td>Avoid[Going to entertainment] [If Lawn work; Lawn work prevent Going to entertainment then] always not Going to entertainment</td>
<td>Lawn work → Going to entertainment</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Table 6.2 gives the number of occurrences of sub-features in features representing the requirements system importance factors. Table 6.3 gives the risk factor. Table 6.4 gives the calculated prioritization and the five requirements are then ordered based on it. Based on the calculated priorities, the five requirements elicited in the section 4.3.2 can be ordered from high to low as follow. Note that the Require and Stop requirements have the same priority.
1. Maintain [Preparing lunch] [If Doing laundry; Preparing lunch sustains Doing laundry then] always Preparing lunch

2. Require[Take medicine] [If Going out for work; Take medicine direct cause Going out for work then] earlier Take medicine

3. Stop[Taking medicine] [If Going to work; Going to work terminates Taking medicine then] no Taking medicine after Going to work

4. Achieve[Putting away groceries] [If Coming from shopping; Coming from shopping direct cause Putting away groceries then] earlier Putting away groceries

5. Avoid[Going to entertainment] [If Lawn work; Lawn work prevent Going to entertainment then] always not Going to entertainment

Table 6.2  *Smart personal assistant requirements system importance factor*

<table>
<thead>
<tr>
<th>Requirement</th>
<th>System Importance Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Require[Take medicine] [If Going out for work; Take medicine direct cause Going out for work then] earlier Take medicine</td>
<td>21</td>
</tr>
<tr>
<td>Achieve[Putting away groceries] [If Coming from shopping; Coming from shopping direct cause Putting away groceries then] earlier Putting away groceries</td>
<td>8</td>
</tr>
<tr>
<td>Maintain [Preparing lunch] [If Doing laundry; Preparing lunch sustains Doing laundry then] always Preparing lunch</td>
<td>17</td>
</tr>
<tr>
<td>Stop [Taking medicine] [If Going to work; Going to work terminates Taking medicine then] no Taking medicine after Going to work</td>
<td>21</td>
</tr>
<tr>
<td>Avoid[Going to entertainment] [If Lawn work; Lawn work prevent Going to entertainment then] always not Going to entertainment</td>
<td>19</td>
</tr>
</tbody>
</table>
Table 6.3  *Smart personal assistant requirements risk factor*

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Qualitative Risk Assessment</th>
<th>Scale(1:Low-Risk, 2:Moderate-Risk, 3:High-Risk)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Require[Take medicine] [If Going out for work; Take medicine direct cause Going out for work then] earlier Take medicine</td>
<td>High-Risk</td>
<td>3</td>
</tr>
<tr>
<td>Achieve[Putting away groceries] [If Coming from shopping; Coming from shopping direct cause Putting away groceries then] earlier Putting away groceries</td>
<td>Moderate-Risk</td>
<td>2</td>
</tr>
<tr>
<td>Maintain [Preparing lunch] [If Doing laundry; Preparing lunch sustains Doing laundry then] always Preparing lunch</td>
<td>Low-Risk</td>
<td>1</td>
</tr>
<tr>
<td>Stop [Taking medicine] [If Going to work; Going to work terminates Taking medicine then] no Taking medicine after Going to work</td>
<td>High-Risk</td>
<td>3</td>
</tr>
<tr>
<td>Avoid[Going to entertainment] [If Lawn work; Lawn work prevent Going to entertainment then] always not Going to entertainment</td>
<td>Moderate-Risk</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 6.4  *Smart personal assistant requirements prioritization*

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Calculated priority with $k_1 = 100$ and $k_2 = 0.1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Require[Take medicine] [If Going out for work; Take medicine direct cause Going out for work then] earlier Take medicine</td>
<td>1.67119846</td>
</tr>
<tr>
<td>Achieve[Putting away groceries] [If Coming from shopping; Coming from shopping direct cause Putting away groceries then] earlier Putting away groceries</td>
<td>0.998068195</td>
</tr>
<tr>
<td>Maintain [Preparing lunch] [If Doing laundry; Preparing lunch sustains Doing laundry then] always Preparing lunch</td>
<td>3.26759941</td>
</tr>
<tr>
<td>Stop [Taking medicine] [If Going to work; Going to work terminates Taking medicine then] no Taking medicine after Going to work</td>
<td>1.67119846</td>
</tr>
<tr>
<td>Avoid[Going to entertainment] [If Lawn work; Lawn work prevent Going to entertainment then] always not Going to entertainment</td>
<td>0.95</td>
</tr>
</tbody>
</table>
6.7 Discussion of proposed situation-oriented requirements evaluation and prioritization approach

Some benefits of the proposed method over the existing methodologies have been observed as follows. The proposed requirements prioritization approach provides knowledge on how each requirement means to a particular individual end-user as well as the system. We believe that this balance between the human viewpoint and the system viewpoint will be a useful resource to the requirements analysts to make better decisions and to gain higher customer satisfaction while not deviating from the system’s developmental milestones. This would make the proposed approach reliable in a wide range of domains. Moreover, the method can also be used to find alternative requirements in order to select the most important and low-risk requirements to perform a particular task.

Despite these significant benefits, the proposed methodology is not without limitations and potential caveats. As with requirements elicitation, the proposed method highly depends on observational data and the unobserved situations and situation-transitions may affect in finding the priorities of the derived requirement. Moreover, the important requirements with inevitable risks may get lower priority and may delay the overall development.

6.8 Literature on requirements evaluation and prioritization

Fast growing software development industry strives to release new products and their enhancements as frequently as possible. In order to reduce the problems starring from resource limitations and time constraints and maximize revenue, many software development companies pay more attention to the evaluation and prioritization of requirements than ever
before. The rest of this section will discuss some of the common techniques as well as some studies in requirements evaluation and prioritization.

### 6.8.1 Direct stakeholder collaboration based approaches

Most of the traditional requirements prioritization techniques allow the stakeholders to prioritize the requirements according to their personal preferences and then form an agreement through identifying conflicts.

The numerical assignment is such a most common approach where stakeholders are requested to place requirements in priority groups [90]. The number of priority groups depends on the software development practice, but three groups: critical, standard, optional are very common [91]. Win-win also is known as Theory-W [92][93] is a prioritization technique that allows each stakeholder to categorize requirements according to the importance and potential risk whereas the Top-Ten requirements approach [88] allows stakeholders to pick their top-ten requirements from a larger set without assigning an internal order between the requirements. In 100-Dollar Test [88] each stakeholder is given 100 imaginary units (such as money, hours, etc.) to distribute among requirements and consider the ratio of the assignment as the scale of the priority. Although most of these techniques are simple and easily manipulated by stakeholders, each has its own limitations. One common problem in these approaches is that most of the stakeholders think that everything is rather critical. For example, a study [94] shows that stakeholders most likely consider 85 percent of the requirements as critical.

The proposed approach does not take direct response from the end-users but instead use the observational data to make an unbiased decision on requirements prioritization. We believe that this will be an effective alternative method to discern the actual critical
requirements of the end-users. In addition, each requirement will be assigned a unique priority which will also be helpful for better decision making. Our proposed approach can be considered as a special case of win-win prioritization technique where the requirements are prioritized based on their relative importance and the potential risks to the stakeholders.

6.8.2 Search based approaches

A well-known search based requirements prioritization approach applies Binary Search Trees (BST) [95] where the prioritization is performed by constructing a binary search tree such that less important requirements are inserted to the left and more important ones to the right. A list of ranked requirements is obtained by using the bubble sort or binary search tree algorithms [88]. This allows stakeholders to compare the relative value of individual requirements and can be used to prioritize relatively large sets of requirements [93]. However, it is believed that original BST ranking is more suitable for a single stakeholder regardless of the sorting algorithm used for ranking since aligning several different stakeholders’ views at the same time might be difficult [79]. In [96] Bebensee et al. introduced the Binary Priority List (BPT) for prioritizing software requirements which is a variation of BST structure. The level of a requirement in the BPT represents its priority level. The top-most level has the highest priority and the priority decreases from top to bottom. Beg et al [97] proposed requirements prioritization technique using B-tree, a self-balancing tree data structure aiming to reduce the number of comparisons between requirements pairs.

In comparison with BST, the situation-transition structure used in the proposed approach is a complex graph with a set of nodes, directed edges and weight values on edges. The unique rank for each requirement was obtained using a graph traversal algorithm developed by modifying the depth-first search algorithms. In other words, the procedure of
the proposed approach is similar BST ranking. However, the proposed approach can be used in both single and multiple end-user domains, and therefore, it is more powerful.

### 6.8.3 Machine learning based approaches

Some researchers are focused on applying data mining and machine learning techniques to the requirements prioritization process in order to improve their effectiveness in using with large software development projects with multiple stakeholders. In [98], Duan et al. proposed a Pirogov approach that uses clustering techniques to place requirements into multiple independent clusters that capture the diverse and complex roles played by individual requirements. Stakeholders determine the relative value of each cluster and weight the importance of each clustering method. An objective function then generates prioritization decisions at the level of the individual requirement.

Tonella et al. [99] proposed an Interactive Genetic Algorithm (IGA) that includes incremental knowledge acquisition and combines it with the existing constraints, such as dependencies and priorities. The proposed approach aims at minimizing the disagreement between a total order of prioritized requirements and the various constraints that are either encoded with the requirements or that are expressed iteratively by the user during the prioritization process. An interactive genetic algorithm was used to achieve such a minimization, taking advantage of interactive input from the user whenever the fitness function cannot be computed precisely based on the information available. The process terminates when a low disagreement is reached, the time out is reached or the allocated elicitation budget has been consumed.

Perini et al. [100] proposed a requirements prioritization method based on Case-Based Ranking (CBRank). The CBRank is originated from a framework that supports
decision making on ordering a set of items which can handle single and multiple stakeholders and different ordering criteria. The prioritization is performed by considering the stakeholder's preferences and the approximated ordering of requirements predicted by the machine learning techniques. Similarly, Babar et al [101] proposed the PHandler, which is an intelligent requirements prioritizing technique that uses artificial neural networks to predict the priority of the requirements.

The algorithm used to generate the situation-transition structure in our proposed approach is a modified version of the Chow-Liu Bayesian structure learning algorithm [28] which is popular in the machine learning area. In other words, the proposed approach also uses machine learning techniques to predict the priority of the requirements based on the observational data.

6.8.4 External factors

Recent research on stakeholder-based requirements prioritizations is focused on enhancing the reliability of the approaches. For example, Ahmad et al [102] discuss limitations of existing requirements prioritization techniques with respect to the geographical distribution of stakeholders and provides a framework to identify important requirements of a product in order to succeed during distributed development. As this recent study suggested, we also believe that consideration of end-users’ human nature and environmental factors such as geographical distribution will improve the quality of the requirement prioritization. Note that, the basic computational unit situation used in our proposed approach is representing information about both aspects.
CHAPTER 7. CONCLUSION

This chapter concludes the dissertation and explains our future research road-map.

7.1 Conclusion

In this work, we present situation-oriented, human-centered approaches to performing Requirements Engineering tasks which are novel because of its focus on human concerns as opposed to the prevalent methods mainly based on business and system perspective. Our major outcomes of this work can be listed as follows.

1. Defining a semi-automated procedure to derive situation-transition-structure using raw sensor data which is a graphical representation of the end-users’ behavioral patterns and the environmental context.

2. Defining a new requirements elicitation technique to analyze the situation-transition-structure and identify the human-centered requirements.

3. Defining a new risk management technique for the Requirements Engineering. This technique first uses the feature models based approach to minimize the gap between human-centered requirements and the actual software functional requirements, then manage the possible risks associated with the human-system interactions.

4. Defining a new requirements prioritization technique which considers the importance of the requirements in both human and system viewpoints as well as the associated risks.

Our ultimate goal of this work is to support the existing Requirements Engineering approaches to provide a better understanding of the end-users’ human-related factors in order to gain higher customer satisfaction and derive safe products.
Although one of the main targets of this work is to automate the proposed approaches as much as possible, at the current stage human involvement is still needed to make critical decisions. We believe that machine learning approach based on existing documentation and log-reports of the software in a similar domain would be a promising solution to the problem. Moreover, the effectiveness of the proposed approaches can be increased with a good collection of observational data gathered from various techniques. The improvements in research fields such as human-activity recognition will have a positive impact on identifying human behaviors as well as the situations; hence can increase the reliability of the proposed Requirements Engineering approaches.

### 7.2 Future work

Our plans for extending the current works lie in several directions as follows.

- We have already established the path to enriching our situation-oriented Requirements Engineering approaches through the introduction of formal specifications [103] since formal requirements specifications are essential in order to maintain the consistency, accuracy, and unambiguousness during the communication between different stages of software development. In this area of future work, we are willing to introduce a new situation-oriented domain-specific system verification and validation procedure in support of human-centered Requirements Engineering which will be intriguing in domains such as socio-technical systems and safety-critical systems. This work will also investigate an effective and efficient approach to dealing with complex constraints and dependences in risk management.
• A situation-oriented human-centered enhancement to the traceability of Requirements Engineering so that it can more effectively encode the relationship between requirements and end-users and how that will be adjusted during the evolution phase.

• As many software systems developed today are of ultra-large scale [104], we are looking forward to improving the effectiveness of situation-transition structure using machine learning techniques and introduce a new situation-transition structure analysis technique to extract useful information. As such the improved situation-transition structure can be extended to provide a comprehensive picture and computational platform in such complex software development. Moreover, we are willing to explore the applicability of data science concepts such as data acquisition, analysis, filtering, and visualization in order to create situation-transition structure using big data [105].
REFERENCES


transactions on Software Engineering and Methodology, 6(1), 1-30.


