

2015

Detection of new intentions from users for software service evolution in human-centric context-aware environments using Conditional Random Fields

Haihua Xie
Iowa State University

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

 Part of the [Computer Sciences Commons](#)

Recommended Citation

Xie, Haihua, "Detection of new intentions from users for software service evolution in human-centric context-aware environments using Conditional Random Fields" (2015). *Graduate Theses and Dissertations*. 14710.
<https://lib.dr.iastate.edu/etd/14710>

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.

Detection of new intentions from users for software service evolution in human-centric context-aware environments using Conditional Random Fields

by

Haihua Xie

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
Morris Chang
David Weiss
Samik Basu
Simanta Mitra

Iowa State University

Ames, Iowa

2015

Copyright © Haihua Xie, 2015. All rights reserved.

TABLE OF CONTENTS

	Page
ACKNOWLEDGMENTS	iv
ABSTRACT	v
CHAPTER 1 INTRODUCTION	1
CHAPTER 2 BACKGROUND AND RELATED WORK	5
2.1 Software Evolution and Requirement Elicitation	5
2.2 The Situ Framework	6
2.3 Conditional Random Fields	8
CHAPTER 3 HUMAN-CENTRIC CONTEXT-AWARE DOMAIN	11
3.1 Sorts of Entities in SiSL	11
3.2 Definition of Action, Desire, Object and Context	15
3.3 Definition and Attributes of Situation	21
3.4 Definition and Attributes of Situation-Sequence and Intention	28
3.5 Situation Pattern and Intention Pattern	31
3.6 Basic Axioms in SiSL	34
CHAPTER 4 DESIRE INFERENCE AND NEW INTENTION DETECTION FOR SYSTEM EVOLUTION USING CRF	41
4.1 Knowledge Base of a Human-centric Context-aware Domain	42
4.2 Desire Inference and New Intention Detection	47
4.3 Using the CRF Method for Desire Inference and New Intention Detection	50
4.4 System Evolution Process Based on New Intention Detection	55
CHAPTER 5 EXPERIMENT ON A RESEARCH LIBRARY SYSTEM	58
5.1 Experiment Platform – the CoRE System	59
5.2 Procedure of an IRB Approved Experiment	61
5.3 Data Collection and Preprocessing	63
5.4 Building the Standard CRF Model	65
5.5 Desire Inference using Hidden Markov Model	71
5.6 Inference Result Analysis in the First-Round Experiment	73
5.7 New Intention Detection Case Study	77
5.8 Validation of System Improvement in the Second-Round Experiment	82
5.9 Summary of the Experiment	85
5.10 Threats to Validity	87

CHAPTER 6 DISCUSSIONS & CONCLUSIONS	93
REFERENCES	95

ACKNOWLEDGMENTS

I would like to thank my advisor and committee chair, Professor Carl K. Chang, and my committee members, David Weiss, Morris Chang, Samik Basu, and Simanta Mitra, for their guidance and support throughout the course of this research.

In addition, I would also like to thank all the colleagues in the Software Engineering Lab for their advice and help on my work. I want to offer my appreciation to my friends and the department faculty and staff for making my time at Iowa State University a wonderful experience. I am indebted to more than 120 participants who contributed to our experiment work. Without whom, this thesis would not have been possible. I would also like to acknowledge the assistance and support of the IRB committee at Iowa State University.

Finally, thanks to my family for their encouragement and days of patience, respect, and love.

ABSTRACT

The capability to accurately and efficiently obtain users' new requirements is critical for software evolution, so that timely improvements can be made to systems to adapt to the rapidly changing environment. However, current software evolution cycles are often undesirably long because the elicitation of new requirements is mostly based on system performance or delayed user feedback and slow-paced manual analysis of requirements engineers. In this thesis, I propose a general methodology that employs Conditional Random Fields (CRF) as the mathematical foundation to provide quantitative exploration of users' new intentions that often indicate their new requirements. My methodology is supposed to be applicable in context-aware software environments, and beneficial for discovering new requirements sooner and considerably shortening software evolution cycles.

First of all, a situation-centric specification language – SiSL, is proposed to formalize the concepts and ontology of the application domains of our methodology. In SiSL, the domain of discourse is divided into five sorts of entities: action, desire, object, situation and situation-sequence. Another two important concepts, context and intention, are defined based on the five basic entities. A set of axioms are proposed to explain the relations among action, context values and desires. Based on the concepts and axioms in SiSL, a domain knowledge base which can completely describe and specify user's behaviors and desires in human-centric context-aware environments can be constructed.

To infer a user's desire based on a peculiar form of observations and a specific detection mechanism for user's new intentions, which may imply new requirements, the Conditional Random Fields (CRF) method is applied as a mathematical foundation to support my research

work. In this thesis, the main part of a CRF model, a set of feature functions, specify the relations between observations (actions and context values) and human internal mental states (desires). To infer user's desires, the CRF model accepts a sequence of observations as the input and calculates the score for each possible sequence-labeling, and outputs the sequence-labeling with the highest score as the inferred desire sequence. By using the CRF method, more accurate desire inference, the precondition for new intention detection, can be achieved compared with other statistical methods.

To detect users' potential new intentions, a CRF model which encodes users' standard behavior patterns should be built as the metrics for outlier detection. The training data for building the standard CRF model are collected from observing user behaviors that are expected to conform to the system design. In the result of desire inference using the CRF model, the divergent behaviors will be labeled with desires in low confidence, and they can be singled out and analyzed for eliciting user's potentially new intentions. Besides the divergent behaviors, user's desire transitions and erroneous behaviors will also be analyzed for detecting new requirements or system drawbacks. The detected potential user's new intention will be verified, analyzed and summarized to generate a formally new intention, which will drive system evolution through modifications or acquiring new functionalities to satisfy the new requirements. An experiment on a research library system has been conducted to demonstrate how to apply our methodology in detection of users' new intentions and driving system evolution. Finally, this thesis discusses the threats to validity for our methodology and experiment.

CHAPTER 1. INTRODUCTION

The goal of software system evolution is to adapt to the ever-changing user requirements and operating environment [1]. Evolvability is an essential quality requirement for most of current software systems [2], [3] because of the ever-evolving and sudden emerging nature of human demands [4] and unpredictable environmental changes for modern-day real-world systems [5]. In order to prolong the productive lifetime of software systems, it is necessary to explicitly address evolvability during the entire software lifecycle [6]. On the contrary, the inability to evolve will cause software system to degrade and become less satisfactory, and eventually obsolete [7]. In practice, the target of cost-effective evolution puts strong demands on software engineers to change software systems on a constant basis with major modifications or enhancements in a timely manner [8].

How to make prompt and effective changes to software systems is a big challenge in software evolution [9]. And all in all, changes shall start with new requirements that specify new user needs or new system environments. Traditionally, these new requirements are elicited based on delayed user feedbacks or business needs and by manual analysis [10], which struggles to keep up with the software evolution pace nowadays. Especially, for many software systems with enormously large user bases, new individual requirements constantly emerge and accumulate. To remain competitive in the business, new software development techniques such as “Agile Methods” are proposed to incrementally develop a working product and deliver it iteratively and frequently (weeks rather than months) [11]. For example, for the android app “k9mail”, its developers release a new version of the application in every two weeks. And such phenomena emerge often in the industry nowadays [60].

To achieve fast evolution, new approaches to elicit new user requirements are much needed. In recent years, most studies on elicitation of new requirements focus on observing historical defects [12], [13] or analyzing users' delayed feedbacks [14], [15], while little work has been done for exploring human intentions that often drive system evolution [16]. Because service environments nowadays are becoming more and more context aware [17], [18], it is possible to observe users' behaviors and environment contexts in real time and analyze users' mental states for acquiring their demands more quickly [19]. To speed up the analysis process, using manual approaches alone is not an option in favor anymore. In fact, as technology advances, applications of mathematical methods in elicitation of new requirements are becoming feasible and even necessary, so that semiautomatic evolution processes can become a serious contender.

A frontier work of human-intention-driven service evolution is Situ [19], a framework that aims to support rapid and iterative service requirements analysis of real-world systems. Situ tries to build a Hidden Markov Model (HMM) to deduce desires of an individual user from given observations (user's actions and environmental context values), and further analyze user's potentially new intentions for driving service evolution. However, Situ encounters difficulties in desire inference because HMM is not able to encode the causal relations among actions, context values and desires, nor the complexity of desire transitions, due to: (1) in a HMM, the current desire is supposed to be independent of neighboring observations [20]. As a result, the relations of a desire and its previous and following observations cannot be reflected. However, in reality, previous context values may influence user's current desire, and following actions and context values may be determined by the current desire; (2) in a HMM, the probabilities of desire transitions are stationary [20]. However, in reality, user's desire change usually depends on

current context values, which may be quite dynamic. And because of the disadvantages mentioned above, Hidden Markov Models are not effective to accurately infer users' desires, not to mention performing detection of new intentions. Therefore, in terms of computability, Situ framework still leaves a lot to be desired.

In this thesis, I propose a general methodology that applies Conditional Random Fields (CRF) as the mathematical foundation to provide quantitative exploration of users' new requirements. CRF is a class of statistical modelling methods often used for encoding known relationships between observations and constructing consistent interpretations [21]. Here CRF is used to build the mathematical model for desire inference, which is to infer users' desires based on runtime observations of their actions and environmental context values. Since intuitively, desire inference can be regarded as labeling an observation sequence with desires, CRF is chosen in our methodology because it is proven to be more accurate for labeling or parsing of sequential data than traditional statistical methods such as HMM and MEMM (Maximum Entropy Markov Model) [22]. Simply speaking, CRF is more suitable for our study because the relations among actions, context values and desires can be better reflected in a CRF model.

This thesis makes the following three contributions:

- (1) A CRF-based method is proposed to effectively infer users' desires based on observations of their actions and relevant environmental context values.
- (2) Three feasible methods are proposed to explore users' new intentions based on the results of desire inference.
- (3) A new software evolution cycle is put forward based on the intention detection methodology that I recommend. To demonstrate and validate my methodology, I conduct a two-round exploration experiment on a user base of 120 participants.

This thesis is organized as follows: Chapter 2 briefly reviews the literatures on system evolution, requirements elicitation, and human desire & intention, and introduces Conditional Random Fields (CRF) and its applications. Chapter 3 describes the attributes of the domains that my methodology can be applied in. Chapter 4 presents the basic concepts and the methodology of building the CRF model and the process of desire inference and new intention detection using the CRF model. Chapter 5 explains the first round experiment by demonstrating through each step in our methodology in details, and presents and analyzes the results, and also briefly presents the second round experiment results, and mainly focuses on evaluating the effectiveness and efficiency of our methodology on enabling software evolution. Chapter 6 concludes the thesis with some speculations.

CHAPTER 2. BACKGROUND AND RELATED WORK

2.1 Software Evolution and Requirement Elicitation

Generally, software evolution refers to the study and management of the process of making changes to software over time [23]. Sometimes, software evolution is defined based on software maintenance, per IEEE's definition [24]: *the process of modifying the software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment*. Lientz and Swanson [25] categorized software maintenance into four different types: adaptive maintenance, perfective maintenance, corrective maintenance and preventive maintenance. The target of software evolution or maintenance is to implement possible major changes to the system to ensure the reliability and flexibility of the system [26]. As a large software system continues to evolve, the complexity of the system will grow [27], meaning that more effective and efficient evolution methods are much needed.

Proposals for change are the drivers for system evolution, and change identification usually continues throughout the system lifetime. The driving forces of the four maintenance activities summarized by Lientz and Swanson [25] are:

- Adaptive maintenance: adapt to changes in the system environment;
- Perfective maintenance: adapt to new user requirements;
- Corrective maintenance: patch system drawbacks;
- Preventive maintenance: prevent problems in the future.

Among the above four problems, the incorporation of new user requirements is the core problem for software evolution and maintenance [9]. Therefore, the capability to accurately and efficiently obtain users' new requirements is a critical issue to be addressed.

Traditional requirements elicitation process can be considered as an interactive mutual learning process between the requirements engineer and the customer [28]. The knowledge of users' requirements can be obtained from interview [43], feedback [10] or observation of customers' activities at their workplace [29]. As users' requirements are usually implicit and unpredictable [30], this process mainly depends on requirements engineers' subjective analysis and judgment, so it is usually time-consuming and results in inaccurate requirements. Oftentimes, a cycle of elicitation, modification, development, and deployment takes a long time to complete, usually several months [19]. Researchers are now facing the steep challenge of shortening such undesirably long evolution cycles. New technologies that can enable automatic or semiautomatic requirements elicitation and analysis are very much desired in order to realize rapid software evolution.

2.2 The Situ Framework

Situ is the first general approach which was proposed for human-intention-driven service evolution in context-aware service environments. Situ is also the first computational framework that allows people to model and detect human intentions by inferring human desires as they are often largely hidden, and capturing the corresponding context values through observations. Equipped with a prediction mechanism, Hidden Markov Model (HMM), in the process of intention detection, Situ is supposed to make instant definition of individualized services at runtime possible, and significantly shorten service evolution cycle.

In order to model and reason human intentions, Situ defines situation as a time-stamped status that includes user's desire, as well as user's actions and relevant context values. It is suggested that the actions performed can be regarded as the external reflection of human internal

mental state, so-called “desire”, and some context value changes are side effects of human actions that are externally observable. Therefore, through observing user’s actions and context values, it is possible to infer user’s desire at each observation time-point using an applicable mathematical mechanism [61]. Furthermore, user’s intention for achieving a certain desire can be obtained by connecting the situations with the same desire as a sequence. Thus, intention is a path in a scenario, similar to that of studies in robotics [62]. Formally, situation at a time t , is defined as a 3-tuple $\{d, A, E\}$, in which d is the inferred or predicted user’s desire, A is a set of actions for achieving a goal which d corresponds to, and E is a set of context values. And an intention is expressed as $I = seq(S_1, S_2, \dots, S_k)$, in which I is an intention and S_1, S_2, \dots, S_k are a temporal sequence of situations threaded through a unique desire d .

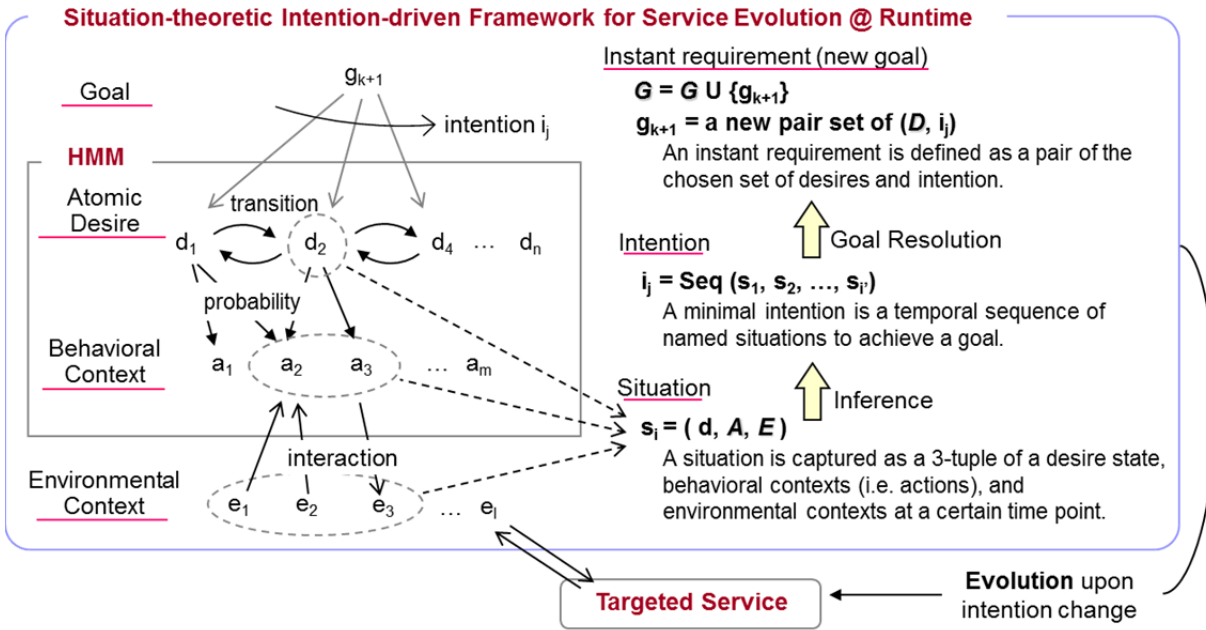


Figure 2.1 *Situ: Framework for service definition with runtime software evolution*

As shown in Figure 2.1, instant requirements can be obtained through goal resolution of the captured intentions. Situ argues that intention change often results in a new goal for the user, requiring the modification of existing features or new pathways to engage new development,

such as creating a new functionality. In this way, the system can be enhanced to satisfy the user's new requirements. Since the software evolution process in Situ is semiautomatic, Situ is considered capable of shortening the software evolution cycles, so that the critical and timely service individualization can be made.

Based on the intention monitoring methodology in Situ, a system is capable of detecting intention changes. An intention change occurs when the inferred intention based on the observed situation sequence is different from any predefined or previous observed intentions. A human intention change will drive the system evolution, as it indicates that the users have new desires, which are also the new requirements for the system. The Hidden Markov Model (HMM) is applied in Situ for intention inferences, and the Viterbi Algorithm (VA), a commonly known and important algorithm for HMM, is used to find the most probable sequence of hidden states based on the visible observations.

As Situ is theoretically breaking a new path for service evolution, many practical problems should be addressed in the future works. For example, how to build an effective model for desire inference and detection of intention changes, how to resolve users' new goals and new requirements, and how to make corresponding changes to the system to adapt to the newly elicited requirements, etc. Furthermore, the Situ framework still needs large-scale experimental validation to show its capability and practicality.

2.3 Conditional Random Fields

CRF (Conditional random fields) are a class of statistical modeling methods used to encode known relationships between observations and construct consistent interpretations [21]. They are often used for labeling or parsing of sequential data, such as natural language text or

biological sequences [22], which are quite similar to observed sequential data in our work. There are several types of CRF models, and the one adopted in this thesis is linear-chain CRF [32].

The general application of the CRF method is sequential labeling, which is to give labels for a sequence of input data. For example, CRF can be used for Part-of-speech (POS) tagging [33], in which the goal is to label each word in a sentence with a POS tag such as ADJECTIVE, ADVERB, NOUN, etc. Before doing this, a set of feature functions are defined to encode the relations between word (data) and POS tag (label). The general format of a feature function is: $f(S, i, l_i, l_{i-1}) = 1$ or 0 , in which:

- S is a sequence of input data;
- i is the ordinal number of current data in S ;
- l_i is the label for the i th observation in S ;
- l_{i-1} is the label for the $(i-1)$ th observation in S ;
- The output is 1 when certain relations specified by the function are satisfied among S , l_i and l_{i-1} , otherwise the output is 0;
- Each feature function is associated with a weight which indicates its labeling reliability.

In order to give the best labeling for an input sequence S , the CRF model calculates the score of every possible sequence-labeling L by adding up the weighted feature functions over all data in the sentence:

$$value(L | S) = \sum_{j=1}^u \sum_{i=1}^v w_j f_j(S, i, l_i, l_{i-1}) \quad (1)$$

(w_j is the weight associated with feature function f_j . u is the number of feature functions in the CRF model and v is the length of S .)

Then, these values are transformed into probabilities $p(L | S)$ between 0 and 1 by exponentiation and normalization:

$$p(L|S) = \frac{\exp[\text{value}(L|S)]}{\sum_{L'} \exp[\text{value}(L'|S)]} = \frac{\exp[\sum_{j=1}^u \sum_{i=1}^v w_j f_j(S, l_i, l_{i-1})]}{\sum_{L'} \exp[\sum_{j=1}^u \sum_{i=1}^v w_j f_j(S, l'_i, l'_{i-1})]} \quad (2)$$

The sequence-labeling L with the largest $p(L | S)$ will be chosen as the labeling for the sentence S . To reduce the computation complexity, the Viterbi Algorithm [34] is applied in computing $p(L | S)$, and the Limited-memory BFGS [35] is a common algorithm used for estimating the weights of feature functions in CRF model training.

Nowadays the CRF method and its extensions or variants are widely used in pattern recognition, machine learning and other domains which deal with structured data [22]. The most popular application of CRF is natural language processing, in which CRF is currently the most advanced technique for many tasks such as named-entity recognition [36], segmenting addresses in Web pages [37], Chinese word segmentation [38], and citation extraction from research papers [39], etc. CRF has also been applied in bioinformatics for protein structure prediction [40] and RNA structural alignment [41]. In computer vision, grid-shaped CRF is used for image segmentation [42]; tree-structured CRF is used for objects recognition [44]. CRF also finds applications in intrusion detection [45] and intent understanding from search behaviors of using search engines [46].

CHAPTER 3. HUMAN-CENTRIC CONTEXT-AWARE DOMAIN

The methodology of new intention detection for system evolution is generally applicable to the human-centric context-aware domains. I use the term “human-centric” to generalize the common nature of various application systems in which humans play a central role in driving system evolution, and the term context-aware to emphasize the physical properties of the system that is sensor-laden for monitoring users’ actions and system status. To formally describe and characterize such domains, a first order language, SiSL [51] (situation-centric specification language), is introduced to specify the entities and essential relations.

3.1 Sorts of Entities in SiSL

SiSL is a first-order language with equality. Standard alphabet of logical symbols: \wedge , \vee , \neg , \exists , \forall , \Rightarrow and \Leftrightarrow , with a full set of connectives and quantifiers in first-order logic are adopted in SiSL, and the alphabet of non-logic symbols in SiSL includes predicate symbols, function symbols, and countable and infinite many individual variable symbols of entities. Besides, some second-logic terms are used in a few formulas of SiSL.

First of all, three assumptions are proposed for stipulating the application scope of SiSL:

Assumption 1: *The domain of discourse of SiSL is a single-agent domain. The agent is believed to be rational, and has desires and corresponding intentions to achieve certain goals. If there are multiple users in the system environment, when constructing the domain for a specific user A , only A ’s own desires and actions will be included, while other users’ desires and actions will not be. In our discussion and inference process, by default, the subject of all actions and desires is the unique agent in the domain.*

Assumption 2: *The unique active agent is in only one application of the system in a time period. And the agent has only one desire at a time instant. Meanwhile, the process of desire transition is memoryless (Markov property), i.e., the conditional probability distribution of future desires depends only upon the present desire, not on the sequence of desires that preceded it.*

Assumption 3: *The domain of discourse of SiSL is a sensor-laden computer application domain. There shall be some sensors deployed by the system to capture the status of the agent, system, and the environment. The set of context values and actions defined in the SiSL-specified domain knowledge base are limited by sensors' observation capability. And the inference of predefined desires and detection of new intentions, are performed based on the observations of actions and context values.*

According to Assumption 1, each SiSL domain has one unique agent in the system, in which the desire and actions at an instant shall belong to the same agent, so that the relationships between the desire and actions can be specified. This assumption rules out the complexity of cases in which one user's desire is directly influenced by other users' actions. E.g. user A changes his/her desire by observing other users' actions. However, other users' influences can be indirectly reflected in relevant context values. Here is an example: Two users A and B are living in a smart home environment. In the domain of user A's smart home system, it is assumed that user B's actions don't have any direct influence on user A's desire; however, they can trigger some context value changes, which may have influence on user A's desire and actions. Such indirect influence can be described by SiSL.

According to Assumption 2, the agent has only one desire at an instant, i.e., the agent doesn't have parallel or concurrent desires. Note that in our discussion, the desires should be consistent with system goals, i.e., they should be relevant to the system domain. Assumption 2 is

also complied with the definition of situation and intention (which will be introduced later in this chapter): there is only one desire in a situation, and intention can be inferred by observing a sequence of situations which share the same desire. Additionally, I assume that the transition of desires should satisfy Markov property, in order to reduce the complexity of the domain, while still keeping it rich enough for desire inference. Later in section 5.6, our experiment evaluation results show that this assumption is in fact appropriate.

Assumption 3 can be considered as a prerequisite of the application of our methodology. Since actions and context values are used as visible states for inferring invisible states (desires), they have to be observable. In our discussion, the term “sensor” represents all general monitoring mechanisms. For example, it can be a hardware sensor device, or a monitoring module hard-coded in the system. It is usually believed that more effective monitoring mechanism can usually help capture more useful data, and the richer the data are, the better our methodology works.

After delimiting the scope of the domains in which our methodology will be applied, the basic concepts in SiSL are defined as follows:

Definition 3.1.1 *There are five sorts of entities for reasoning about users’ behaviors and internal mental states in software system domains – action, desire, situation, situation-sequence and object. Any entity in the domain must belong to exactly one of the five sorts.*

Countable and infinite many individual variable symbols can be defined for each sort of entities. We shall use a , d , o , s and q , with subscripts and superscripts, for variables of sort action, desire, object, situation and situation-sequence respectively. For example, variables of sort *action* can be denoted as a_1, a_2, \dots, a_n , and variables of sort *situation-sequence* can be denoted as q_1, q_2, \dots, q_m .

The basic relations of the five sorts of entities are given Table 3.1.

Table 3.1 Lexicon for Basic Theories in the SiSL Ontology

Entities	<i>Action(a)</i>	<i>a is an action</i>
	<i>Desire(d)</i>	<i>d is a desire</i>
	<i>Object(t)</i>	<i>o is an object</i>
	<i>Situation(s)</i>	<i>s is a situation</i>
	<i>SitSeq(q)</i>	<i>q is a situation-sequence</i>
Situation	<i>ActionIn(a, s)</i>	<i>a is a user's action in situation s</i>
	<i>DesireIn(d, s)</i>	<i>d is the user's desire in situation s</i>
	<i>Before(s₁, s₂)</i>	<i>situation s₁ happens before situation s₂</i>
	<i>After(s₁, s₂)</i>	<i>situation s₁ happens after situation s₂</i>
	<i>StaticSituation(s)</i>	<i>s is a static situation</i>
	<i>SatisfiedSituation(s)</i>	<i>s is a satisfied situation</i>
Situation-sequence	<i>SituationIn(s, q)</i>	<i>s is a situation in situation-sequence q</i>
	<i>Initial(s, q)</i>	<i>s is the initial situation of situation-sequence q</i>
	<i>Final(s, q)</i>	<i>s is the final situation of situation-sequence q</i>
	<i>Prev(s₁, s₂, q)</i>	<i>s₁ is the previous situation of s₂ in situation-sequence q</i>
	<i>Next(s₁, s₂, q)</i>	<i>s₁ is the next situation of s₂ in situation-sequence q</i>

Sorts Foundation Axiom: The attribute that any entity in the domain belongs to exactly one of the five sorts can be expressed by the following axiom:

$$\forall x. \{Action(x) \vee Desire(x) \vee Object(x) \vee Situation(x) \vee SitSeq(x)\} \wedge \neg \exists x. \{[Action(x) \wedge Desire(x)] \vee [Action(x) \wedge Object(x)] \vee [Action(x) \wedge Situation(x)] \vee [Action(x) \wedge SitSeq(x)] \vee [Desire(x) \wedge Object(x)] \vee [Desire(x) \wedge Situation(x)] \vee [Desire(x) \wedge SitSeq(x)] \vee [Object(x) \wedge Situation(x)] \vee [Object(x) \wedge SitSeq(x)] \vee [Situation(x) \wedge SitSeq(x)]\}.$$

Sub-sort Entity: Sub-sorts of the five basic-sort entities can also be defined. A sub-sort is defined as a predicate, and its super-sort is specified using an axiom. For example:

Predicate *Button*: $object \rightarrow True \cup False$, represents a sub-sort of the sort *object*: *Button*. And the axiom: $\forall x. Button(x) \Rightarrow Object(x)$ specifies that the super-sort of *Button* is *object*. $Button(Btn_Submit)$ represents that *Btn_Submit* is a button, and *Btn_Submit* is also an object.

3.2 Definition of Action, Desire, Object and Context

Among the five basic sorts of entities, situation is the core concept, and other sorts of entities can be specified based on situation. The relations between situation and action, desire, object, situation-sequence respectively can be drawn as Figure 3.1.

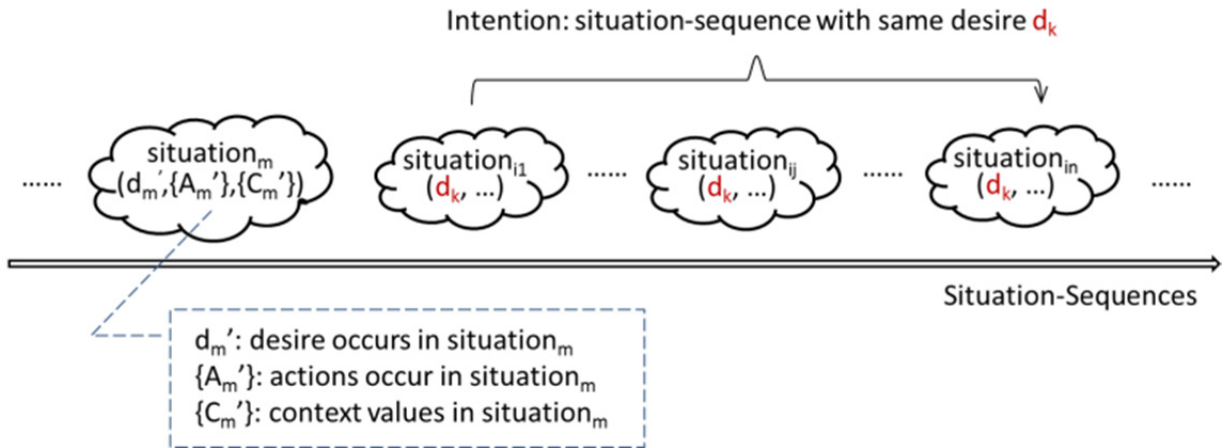


Figure 3.1 Relations of different entities in the domain.

The formal definition and description of concepts in SiSL are given as follows:

3.2.1 Definition and Attributes of Action

Action: agent's behavior performed in the system environment, particularly their operations on the system interface. We can denote actions using action functions, while we can also declare constants or variables for actions. The formal definition of action function in SiSL is:

Definition 3.2.1 For each $n \geq 0$, a finite number of function symbols with arity n , and sorts $(\text{object})^n \rightarrow \text{action}$. These functions are called action functions.

An action function takes objects as the input and an action as the output. The domain of an action function can also be a sub-sort of object, and this sub-sort should be specified when defining the action function. Some examples of action functions are:

- $\text{act_clickMenuoption}: \text{Menuoption} \rightarrow \text{action}$. The sort Menuoption is the set: $\{\text{Menuopt_Home}, \text{Menuopt_Abstract}, \text{Menuopt_File}, \text{Menuopt_Reviewer}\}$;
- $\text{act_inputNumber}: \text{Number} \rightarrow \text{action}$. The sort Number is the set: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$;

According to the above two definitions, $\text{act_clickMenuoption}(\text{Menuopt_Home})$ represents the action *click menu option Home*, and $\text{act_inputNumber}(1)$ represents the action *input number 1*. It's not necessary to define constant symbols for these actions.

There are two benefits for defining actions as action functions instead of constants:

(1) Classify Actions

Some actions can be classified to be a group. For example, *click menu option 'x'* and *click button 'y'* can be classified into a group of *click* action, and *input number 0* and *input character 'a'* can be classified into a group of *input* action. For the same class of actions, the corresponding action-triggered context value changes or reactions are usually similar, and sometimes they can be handled in the same way. For example, in the CRF computational framework, there is a feature function: $f(s, i, l_i, l_{i-1})$ which states that if the user's action is *input (something)* and one of the context values is: the current page is an abstract submission form, the use's desire is *submit an abstract*. This feature function is easy to be defined if there is an action function $\text{input}(o)$ because the action *input* can be easily identified.

(2) Indicate objects involved in an action

Sometimes we need to analyze the influences of an action on the objects, in this way it's important to indicate objects involved in an action. In action functions the action and objects are split so that the objects can be easily indicated. This attribute is useful for defining the action determinisms.

Action Occurrence: actions are general definitions, and they cannot express the real-time behaviors. When we want to present an action is being performed in a situation, we will write a formula as: $ActionIn(a(x_1, x_2), s)$, in which $a(x_1, x_2)$ is an action and s is a situation.

Action occurrence represents the occurrence of an action, and it is denoted as $ActionIn(a, s)$, which is a predicate. In each situation, each action has a corresponding action occurrence. If an action is happening in the situation, its action occurrence is true, otherwise it is false. Therefore, an action doesn't have situations involved, while action occurrence is a status in a situation.

3.2.2 Definition and Attributes of Desire

Desire: the condition or status of entities that the agent would like to achieve. The desires drive users to perform actions to achieve certain goals. We usually denote desires using constants or variables, and we can also define desire functions. The formal definition of desire function is:

Definition 3.2.2 *For each $n \geq 0$, a finite number of function symbols with arity n , and sorts $(object)^n \rightarrow desire$. These functions are called desire functions.*

A desire function takes sort object as inputs and takes sort desire as outputs. The domain of a desire function can also be a sub-sort of object, and it must be specified when defining the desire functions. Some example of the desire functions are:

- *login: Reviewer* \rightarrow *desire*. The sort *Reviewer* is the set of all reviewer accounts in the database;
- *download: Paper* \rightarrow *desire*. The sort *Paper* is the set of all paper IDs in the database;

According to the above two definitions, *login(John@gmail.com)* represents the desire *login reviewer account John@gmail.com*, and *download(P₀₁)* represents the desire *download paper P₀₁*. It's not necessary to define constant symbols for these desires.

Similar to action functions, there are two benefits for defining desire functions instead of defining constants for desires:

(1) Classify desires

Some desires can be classified to be a group of desires. For example, *submit paper P₀₁* and *submit paper P₀₂* can be classified as a group of *submit* desires, and *upload file x* and *upload file y* can be classified as a group of *upload* desires. The desires in a group have similar attributes and can be handled in the similar way.

(2) Indicate objects involved in a desire

Because desires are usually expressed as predicates on objects, it is essential to indicate objects involved in a desire so that the relations of objects can be easily described. Desire functions are useful for defining desire satisfaction axioms.

Desire Occurrence: desires are general definitions, and they cannot express the real-time human mental states. When we want to describe user's desire in a situation, we will write a formula as: *DesireIn(d(x₁, x₂), s)*, in which *d(x₁, x₂)* is a desire and *s* is a situation.

Desire occurrence represents the occurrence of a desire, and it is denoted as *DesireIn(d, s)*, which is a predicate. In each situation, each desire has a corresponding desire occurrence. If a

desire is happening in the situation, its desire occurrence is true, otherwise it is false. Therefore, a desire doesn't have situation involved, while desire occurrence is a status in a situation.

3.2.3 Definition and Attributes of Object

Object: any entity other than desire, action, situation and situation-sequence in the domain, such as *user*, *key 'x'*, *button "Submit"*, etc. The object can be created or destroyed in a situation. For example, P_{10} is created when a paper is submitted and its ID in the database is 10. And when the paper is deleted, the object is destroyed.

The object set is a catch-all set. It normally refers to the concrete entities, whereas it can also refer to the abstract entities, such as the numbers. We use sub-sorts to define specific categories in the sort object. For example, we can define predicate Integer to represent the integer set. Some concrete objects have lifespan which is the duration between its created situation and destroyed situation. An object can also exist forever, and its lifespan is *sit-* to *sit+*.

As presented before, user's desires are inferred based on observations of user's actions and context values in the system environment. Below we give the definition of contexts.

3.2.4 Definition and Attributes of Context

Context: any information that is used to characterize the status of objects in the domain of discourse. It can be regarded as relations between objects and situations. There are two kinds of contexts as follows:

- Functional Context: for each $n \geq 0$, a finite number of functional symbols of sorts $(object)^n \times situation \rightarrow object$. They denote contexts such as: *titleOfPaper(x, s_n)*, *numOfSubmittedPapers(s_n)*.

- Predicate Context: for each $n \geq 0$, a finite number of predicate symbols of sorts $(object)^n \times situation$. They denote relations such as: $CurrentPage(Page_Home, s_0)$, $LoggedOn(x, s_1)$.

Notice that a context always takes sort situation as an argument (and always the last argument), so contexts are situation dependent. Apart from situation, only terms of sort object are arguments of a context, so contexts are status of objects in situations.

The action functions and desire functions are not contexts, although they are also used to describe the instant status of the domain. There are two reasons:

- action functions and desire functions do not describe the status of the system, but the user;
- action functions and desire functions do not take situation as a parameter;

The set of contexts consists of functional contexts and predicate contexts. A functional context is also a function, and a predicate context is a predicate. We can simply denote the relations of contexts, predicates and functions as follows:

$$\{functional\ context\} \cup \{predicate\ context\} = \{context\}$$

$$\{context\} \cap \{function\} = \{functional\ context\}$$

$$\{context\} \cap \{predicate\} = \{predicate\ context\}$$

Functions and predicates in SiSL are classified into several kinds. The classifications of predicates are:

- predicate contexts;
- domain-independent predicates: domain independent relations, whose values are independent from the domains, such as $ProgrammingLanguage(PHP)$, $ValidEmailAddress(Michael@gmail.com)$;

- domain-specific predicates: facts in the domain, whose values are dependent on the domains, such as: *LanguageOf(MyReview, PHP)*, *AcceptedFileType(PDF, MyReview)*;

The classifications of functions are:

- functional contexts;
- action functions;
- desire functions;
- domain-independent functions: such as *sumof(1, 2)*, *distance(NewYork, Chicago)*;
- domain-specific functions: such as *userTypes(MyReview)*, *url(homepage)*.

Context values are discrete. In practice, the values of some contexts, such as the temperature, are infinite and keep changing all the time. But in SiSL, the value set of each context is finite, so that its runtime value can be captured. For example, to measure the temperature we use a thermometer which has finite calibrations.

3.3 Definition and Attributes of Situation

Based on the introduction of the above concepts, I will give the definition of situation as follows:

Definition 3.3.1 *A situation is an instant status of the user and the system environment, including the user's behaviors and internal mental states, and the status of objects in the environment.*

The basic attributes of situation in SiSL are:

(1) Situation is a sort of entity in SiSL, and it can only be denoted as a constant or a variable. As a comparison, situation is denoted as a triple $\{d, A, E\}_t$ in Situ;

(2) The implication of situation is the instant domain status. The following statuses in the domain are changing over time and needed to be determined at an instant (as assumptions of SiSL, only one active user in the domain and the user only has one desire at an instant): the user's desire; the user's actions; the value of each context. Therefore, the implication of situation in SiSL is same to that in Situ;

(3) The implication of a situation is completely described when the values of desire occurrences, action occurrences and contexts at an instant are all determined. Because situation is a sort of entity, it cannot directly describe the statuses of the user and the environment. For instead, three kinds of predicates in SiSL are used to express the implication of a situation:

- *DesireIn*(d, s), desire occurrence: because the user only has one desire in a situation, there is only one desire occurrence whose value is true in a situation. E.g., if the user's desire is d_1 in situation s , there is: $DesireIn(d_1, s) \wedge \neg DesireIn(d_2, s) \wedge \neg DesireIn(d_3, s) \dots$;
- *ActionIn*(a, s), action occurrence: there could be more than one action occurrences are true in a situation;
- Context: the statuses or relations of objects in a situation.

In sum, in order to completely describe a situation, the values of following statuses need to be determined:

- All the desire occurrences: we need to determine which desire occurrence is true, and set the rest to be false;
- All the action occurrences: some of them can be determined based on the observation results of the deployed sensors;

- All the context values: some of them are determined based on the observation results of the deployed sensors;

3.3.1 Advantages of the definition of situation in SiSL

The definition of situation is critical in SiSL because it is the core concept and other concepts are defined based on situation. Followings are some possible ways to define situation:

(1) Bijective function on time: $time\text{-}point \rightarrow situation$. In this way, there is a situation at every time-point, and every situation is unique in the domain. This definition brings two difficulties:

- An intention is usually defined as a sequence of situations for achieving a goal. If situations are continuous, the intention should be redefined to exclude unimportant and redundant situations;
- Because continuously observing situations is impossible, the relations of the “real world” and the “observed world” must be specified, i.e., how to reason all the situations, including the unobserved situations, based on the observed situations.

(2) Surjective function on time: $sit(t) = \{d(t), A(t), C(t)\}$, in which $d(t)$ is the user’s desire at time-point t , $A(t)$ is the set of all actions happen at t , $C(t)$ is the set of values of all contexts at t . In this definition, the situations are also continuous but two situations can be the same if the desires, actions and context values at the time-points of the two situations are the same. Some problems with this definition:

- $d(t)$, $A(t)$ and $C(t)$ should be specified first. E.g., the action at time-point t is actually the action occurrence at time-point t , thus the action occurrence should be clearly defined. Similarly, the desire occurrence should also be defined;

- In this definition the desire occurrences and action occurrences are defined on time-point. However, in SiSL, desire occurrences and action occurrences are defined on situation. E.g., $DesireIn(d, s)$ means the user's desire in situation s is d .

(3) A situation is an instant status when an action effects, a context value changes or a desire arises. In this way, the situations are discrete, and they are bijections on action effect, context value changes or desire arising. Followings are the problems of this definition:

- The action effect and desire arising should be clearly defined. Two new sorts: $action_effect$ and $desire_arising$ are necessary;
- The action effects are hard to specify, because action effects are often referred to context value changes, but the effects on context values of every action are complex. Besides, the end of a desire should also be defined as a situation. In this way, the theory is more complicated.

(4) Situations are functions on action, context value and desire. E.g., a new situation can be defined as: $sit_2 = do(action_1, sit_1)$; $sit_2 = changeTo(cont_1(value_1, s_1), value_2)$; $sit_2 = arise(desire_1, sit_1)$. The problem of this definition is:

- There will be no concurrencies of actions, which sometimes are important for determining desires. A possible solution can be: add new functions like: $sit_3 = end(action_1, sit_2)$. However, the theory will be more complicated and hard to be specified.

(5) Situations are functions on snapshots, which represent the instants of an observation in practice. An observation can be action-triggered, context value change triggered or periodically taken. There are some problems with this definition:

- Impossible to describe the relations between two consecutive situations, since they could have no relations and some action occurrence and context value changes might be missed;
- A solution can be: specify time-point as: the instant of each action effects, each context value changes or each desire rises. Then this definition is same to (3).

In SiSL, the definition of situation is similar to definition (1), while the time attribute is hidden. To deal with the problems which are given in (1), the following changes are made:

- The definition of intention is slightly changed to exclude unimportant and redundant continuous situations;
- In practice we don't capture all the situations, but only the situations when significant action occurs and context value changes. We analyze the "real world" based on the "observed word" through using the CRF computational framework.

3.3.2 Attributes of situation in SiSL

Some attributes of situation are presented below:

- (1) Single desire and multi actions in a situation

According to **Assumption 2**, a situation can contain no more than one desire, while it can contain more than one action, which are performed simultaneously. There are:

$$(\forall s, d_1, d_2). \{DesireIn(d_1, s) \wedge d_1 \neq d_2 \implies \neg DesireIn(d_2, s)\}$$

$$(\exists s, a_1, a_2). \{a_1 \neq a_2 \wedge ActionIn(a_1, s) \wedge ActionIn(a_2, s)\}$$

An example of more than one action being performed in one situation is: the user presses button "z" with his right hand and presses button "Shift" with his left hand, and his desire is inputting "Z".

(2) Situations are continuous

In previous sections, I already stated that the situations are continuous. To deal with the continuity and infinity of situations, the following predicates are proposed:

- $Dissimilar(sit_1, sit_2)$: at least one action occurrence, context value or desire occurrence is different in situation sit_1 and sit_2 , whereas two situations are always different;
- $Transition(sit_1, sit_2)$: sit_1 and sit_2 are dissimilar and no other situation between them is dissimilar to both of them, and sit_1 happens before sit_2 . There is no situation could transit to two different situations:

$$\neg \exists s_1, s_2, s_3. Trans(s_1, s_2) \wedge Trans(s_1, s_3)$$

- $Reach(sit_1, sit_2)$: situation reaching denotes the reaching from a situation to another:
 $(\forall s_1, s_2). Reach(s_1, s_2) \Leftrightarrow Trans(s_1, s_2) \vee \exists s_3. (Reach(s_1, s_3) \wedge Reach(s_3, s_2))$
- $TransitionPoint(sit)$: sit is a situation which will become *Dissimilar* immediately;
- $ConsistentPeriod(sitt_1, sitt_2)$: $[sitt_1, sitt_2]$ is a period in which any two situations are not *Dissimilar*.

The following diagram shows the continuity and transition of situations:

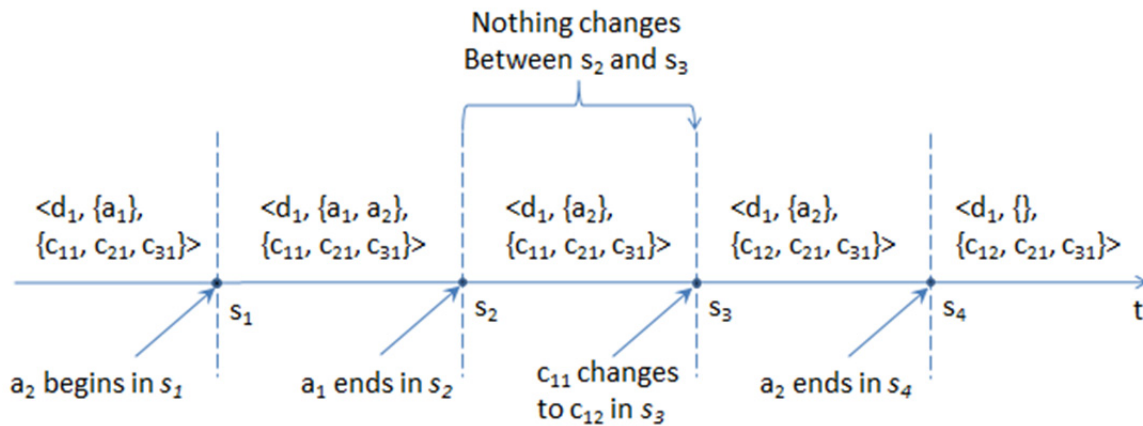


Figure 3.2 Continuity and transition of situations in SiSL

(3) Time Attribute

In SiSL, there is no entity for representing time, but situation has the time attributes. There are two predicates for specifying the time attribute of situation:

- $Before(s_1, s_2)$: s_1 happens before s_2 ;
- $After(s_1, s_2)$: s_1 happens after s_2 .

The time attribute of situation has the following aspects:

1) Ordering: the sequence of *situations* is linearly ordered, forwards to the future and backwards into the past. There is an axiom to specify the ordering attribute of situation in SiSL:

$$(\forall s_1, s_2). Situation(s_1) \wedge Situation(s_2) \wedge s_1 \neq s_2 \Leftrightarrow Before(s_1, s_2) \vee After(s_1, s_2)$$

2) Infinity: the situation line is infinite, with two endpoints: *sit-* and *sit+*;

3) Density: the set of situations is dense, i.e., between any two situations there are infinite situations:

$$(\forall s_1, s_2). Before(s_1, s_2) \Leftrightarrow \exists s_3. Before(s_1, s_3) \wedge Before(s_3, s_2)$$

4) Instant: In a situation, the domain status, including action occurrence, desire occurrence and context values, stagnates and nothing changes;

5) Abstraction and Instances: the time attribute contained in situations does not reflect the real time, but its abstraction. In different domains, the time formats can be different.

For example, if a domain uses *GMT* as the real-time format, a function $gmt(sit)$ can be defined to get the *GMT* time in a situation, and a time theory can also be built to specify the duration and ordering of situations. For example, an axiom in the *GMT time theory* is:

$$(\forall s_1, s_2, d). d = duration(s_1, s_2) \Leftrightarrow$$

$$gmt(s_2) = time_add(gmt(s_1), d) \wedge Situation(s_1) \wedge Situation(s_2) \wedge Object(d).$$

In the above formula, $duration(t_1, t_2)$ is a function calculates the duration between two GMT time duration between two *situations* s_1 and s_2 and $time_add(gmt(s_1), d)$ is a function adds duration d to the GMT time $gmt(s_1)$ and gets a new GMT *time*. The function $time_add$ can be specified using other axioms. We need a complete *time theory* which can be referred to the *time theory* in Process Specification Language (*PSL*).

(4) Static situation

A static situation is a situation without any action. A sub-sort of situation: *StaticSituation* is defined to denote the static situations. There is:

$$StaticSituation(s) \Leftrightarrow Situation(s) \wedge \forall a. \neg ActionIn(a, s)$$

Because the context values changes are not necessarily directly triggered by the actions, so sometimes there are some situations without any action, and these situations are static situations.

(5) Satisfied situation

A satisfied situation is a situation in which the desire is satisfied. Situation s is called a satisfied situation if only if the desire in this situation is satisfied. A sub-sort of situation: *SatisfiedSituation* is defined as:

$$SatisfiedSituation(s) \Leftrightarrow Satisfied(d(s), s)$$

$Satisfied(d(s), s)$ can be decided to be true or not based on the desire satisfaction axioms, which will be presented later.

3.4 Definition and Attributes of Situation-Sequence and Intention

In this section, I give the formal definition of situation-sequence and intention. Situation-sequence is a sort of entity in SiSL, and an intention is a special situation-sequence with specific attributes.

3.4.1 Definition and Attributes of Situation-Sequence

Definition 3.4.1 A situation-sequence is a chronological sequence of situations.

I use a special notation ‘ \sim ’ to denote a situation sequence. The definition is:

Definition 3.4.2 A function symbol $\sim: (situation \vee situation\text{-}sequence) \times (situation \vee situation\text{-}sequence) \rightarrow (situation\text{-}sequence \vee False)$, which denotes the formation of situation-sequences.

According to the above definition, a situation-sequence can be denoted as:

$situation\text{-}sequence \equiv situation \sim situation$

$\vee situation \sim situation\text{-}sequence$

$\vee situation\text{-}sequence \sim situation$

$\vee situation\text{-}sequence \sim situation\text{-}sequence$

The function \sim will return *False* in the following cases: (1) $sit_1 \sim sit_2 = False$ if sit_1 occurs after sit_2 ; (2) $sit_1 \sim sitseq_1 = False$ if sit_1 occurs after at least one situation in $sitseq_1$; (3) $sitseq_1 \sim sit_1 = False$ if sit_1 occurs before at least one situation in $sitseq_1$; (4) $sitseq_1 \sim sitseq_2 = False$ if the last situation in $sitseq_1$ occurs after the first situation in $sitseq_2$.

There are some attributes of a situation-sequence as below:

I. Basic axioms: there are some basic axioms for an intention as follows:

$$(\forall q, s).Initial(s, q) \Rightarrow \forall s_1. \neg Prev(s_1, s, q)$$

$$(\forall q, s).Final(s, q) \Rightarrow \forall s_1. \neg Next(s_1, s, q)$$

$$(\forall q, s_1, s_2).Next(s_1, s_2, q) \Leftrightarrow Prev(s_2, s_1, q)$$

$$(\forall q, s). \{ SituationIn(s, q) \wedge \neg Initial(s, q) \wedge \neg Final(s, q) \Rightarrow \exists (s_1, s_2). Prev(s_1, s, q) \wedge Next(s_2, s, q) \}$$

II. Temporally ordered: the situations in a situation-sequence are temporally ordered.

For any two consecutive situations in a situation-sequence, the latter situation occurs after the former situation:

$$(\forall q, s_1, s_2).Prev(s_2, s_1, q) \Rightarrow Before(s_2, s_1)$$

$$(\forall q, s_1, s_2).Next(s_2, s_1, q) \Rightarrow After(s_2, s_1)$$

III. Reachability: All the situation reaching is possible:

$$(\forall q, s_1, s_2).Next(s_2, s_1, q) \Rightarrow Reach(s_1, s_2)$$

Intuitively, a situation-sequence should not be defined as a sort, but a set of situations. However, it will be hard to describe the attributes of situation-sequence if it is as set, because the first (second) order logic doesn't allow the definitions of functions and predicates on sets. For example, $Final(s, q)$ is not allowed if q is a set of situations.

3.4.2 Definition and Attributes of Intention

Definition 3.4.3 *An intention is a chronological sequence of situations which tends to satisfy a certain user's desire.*

An intention is defined as a special situation-sequence with specific attributes in SiSL. An intention q has the following attributes:

- All the situations in the sequence have a same desire: $SameDesire(q)$;
- Any pair of adjacent situations in the sequence is dissimilar: $DissimilarSeq(q)$;
 - Exact one situation in a *ConsistentPeriod* is chosen in the sequence, and it is not necessary the situation at the *Transitionpoint*.
- The user's desire is finally satisfied, but it's not satisfied in the process: $SatisfiedSeq(q)$;

- The final situation in the intention is a satisfied situation but not any situation before the final situation is a satisfied situation.
- All relevant situations are involved: *CompleteSeq(q)*.
 - For any pair of adjacent situations $(s_i, s_i + 1)$ in an intention, there is no situation which is between s_i and $s_i + 1$, dissimilar to s_i and $s_i + 1$, and has the same desire with s_i and $s_i + 1$.

Based on the above attributes, there is:

$$Intention(q) \Leftrightarrow SameDesire(q) \wedge SatisfiedSeq(q) \wedge CompleteSeq(q) \wedge DissimilarSeq(q)$$

In practice, we can only capture the *sub-intention* of an *intention*, and a *sub-intention* is a *sub-situation-sequence* of an *intention* with the same end *situation*.

3.5 Situation Pattern and Intention Pattern

In this section, I will introduce the formal definitions and attributes of situation pattern and intention pattern.

3.5.1 Definition and Attributes of Situation Pattern

Situation is the catch-all status of the domain and it can only be generated at runtime. To analyze the properties of situations and relations between situations, situation pattern is defined to describe the common attributes of possibly appearing situations. The definition of situation pattern is:

Definition 3.5.1 *A situation pattern inscribes the common attributes of a class of situations which possibly appear in practice. It must include the user's current desire, and the*

actions determined by the desire associated with the context values. The irrelevant actions and contexts should not be included in a situation pattern.

The situation patterns are expressed as unary predicates on situation. For example:

$$SP_I(s) \Leftrightarrow DesireIn(Des_SubmitAbstract, s) \wedge ActionIn(act_click(Menuopt_Abstract), s)$$

The attributes of situation pattern and the differences between situation and situation pattern are:

- (1) Situation pattern describes the attributes of situations, so different situations can have the same situation pattern, while a situation is unique in the domain;
- (2) Situation pattern doesn't have the time attribute, and it can be predefined in the domain knowledge base, while situation can only be generated at runtime;
- (3) Situation pattern describes the significant attributes of situations and discard some unimportant or irrelevant attributes, while situation is the catch-all status of the domain;
- (4) The desire, actions and context values involved in a situation pattern must satisfy the action determinisms, i.e., the actions are taken from a set of possible actions determined by the desire and context values;

In the domain knowledge base, there are several predefined situation patterns, which are expressed as predicates. In practice, situations are observed and their situation patterns will be determined.

3.5.2 Definition and Attributes of Intention Pattern

Intention pattern: an intention pattern can be viewed as a standard/designed sequence of situations for achieving a desire, while the runtime intentions are usually more complicated

because they may contain errors, repeats, interruptions, restarts, etc. The attributes of intention pattern are as follows:

(1) The intention pattern is defined based on situation patterns, i.e., the attributes of situations contained in an intention pattern is described using situation pattern;

(2) The last situation in an intention pattern must be a satisfied situation because the desire must be satisfied in the end of the intention. The situation pattern of the last situation must contain the attribute: satisfied, and other necessary attributes for specifying the intention;

(3) Intention patterns represent the standard process of users' operations on the system to achieve certain desire. Intention patterns are usually derived from the use case scenarios of a software system, while the practical intentions may contain errors, interrupts, repeats or restarts;

(4) Intention patterns reflect the design of the system because they mainly describe user's operations on the system.

(5) Intention patterns are defined as predicates, and specified using axioms. For example:

$$\begin{aligned}
 IP_1(i) &\Leftrightarrow SP_1(initialOf(i)) \wedge SP_6(finalOf(i)) \wedge \\
 &\{SP_2(nextOf(initialOf(i), i)) \vee SP_3(nextOf(initialOf(i), i))\} \wedge \\
 &\{\forall s.SP_2(s) \wedge SituationIn(s, i) \Rightarrow SP_2(prevOf(s, i)) \vee SP_3(prevOf(s, i))\} \wedge \\
 &\{\forall s.SP_4(s) \wedge SituationIn(s, i) \Rightarrow SP_2(prevOf(s, i)) \vee SP_5(prevOf(s, i))\} \wedge \\
 &\{\forall s.SP_5(s) \wedge SituationIn(s, i) \Rightarrow SP_4(prevOf(s, i)) \vee SP_5(prevOf(s, i))\} \wedge \\
 &SP_5(prevOf(final(i), i)).
 \end{aligned}$$

IP_1 is an intention pattern predicate. The right side of the above formula specifies the attributes of the intention pattern. Any situation sequence which satisfies the attributes can be viewed to be IP_1 .

3.6 Basic Axioms in SiSL

In this section, I will introduce the basic axioms in SiSL. The relationships among different entities in SiSL are significant for constructing the domain knowledge base. The three kinds of relations are fundamental in SiSL:

- The relation between action and desire: because user's actions are mainly determined by his desire and the current context values, and there is only one active user during a time period in the domain (assumption 1), the relation between action and desire can be specified using certain axioms, which are called action determinism in SiSL;
- The relation between action and context: context value changes are side effects of human actions that are externally observable. How to reach certain context values based on actions are specified by context value determinisms in SiSL;
- The relation between context and desire: the conditions (a set of context values) under which a desire is satisfied are specified by desire satisfaction axioms in SiSL.

The above three kinds of axioms are introduced as below:

(1) Action Determinism: *Action determinism: The set of possible actions in any situation is determined by the desire and context values in the current situation. The **action determinism** is represented as the following formula:*

$$ActionIn(a(x_1, \dots, x_n), s) \Rightarrow \Delta_a(x_1, \dots, x_n, desireOf(s), s))$$

$\Delta_a(x_1, \dots, x_n, desireOf(s), s)$ is a formula consists of predicates, connected by logical symbols, and $desireOf(s)$ is a function returns the desire in situation s . $\Delta_a(x_1, \dots, x_n, desireOf(s), s)$ is a sentence, with no free variable.

Action Determinisms are used to encode the relations between actions and desires, as well as context values, and they mainly describe the user aspect of the domain. Action determinisms usually describe user's rational behaviors. An example of action determinisms is:

$$\begin{aligned} &ActionIn(act_click(Btn_Admin), s) \Rightarrow \\ &\quad \{desireOf(s) = Des_CheckSystemInfo \wedge \\ &\quad \quad curPageOf(s) \neq (Page_AdminLogin \vee Page_AdminMenus)\} \vee \\ &\quad \quad desireOf(s) = Des_TestSystem \end{aligned}$$

The explanation of the above formula is: when the user's action is "click menu option *Btn_Admin*", his desire is supposed to be one of the following cases: "check the system information (if the current page is not '*admin login*' or '*admin menus*') or "test the system". Notice that in the formulas, the connection symbol is " \Rightarrow " not " \Leftrightarrow " because " \Leftrightarrow " that means the user will perform the action at every time-point when the user has the desire.

The action determinisms are very useful for defining the abnormal behaviors in the CRF models because they are usually defining the normal cases.

(2) Context value determinism: *The values of some contexts in future situations are determined by the actions and context values in the current situation, while other context values are not influenced by any action, depending on the nature of those actions and contexts.* I define the context value determinisms instead of the action affect axioms because:

- 1) Some context values are not affected by the actions directly, such as the temperature, so the action affect axioms cannot comprehensively explain the context value changes;
- 2) Some context value changes can be triggered by different actions. It's better to put these action affects together;

- 3) The actions may have no impact on some context values. These un-changes should be specified.

Therefore, I define the context value determinisms instead of the action affect axioms. In the context value determinisms, the following information is included:

- 1) The context value changes triggered by actions;
- 2) The context value changes not triggered by actions;
- 3) Context value un-changes.

The following assumptions are preconditions for defining context value determinisms:

- 1) The value set of a context is finite and discrete, but the set of situations is infinite and continuous;
- 2) Context value changes complete instantaneously, i.e., if the value of a context c changes from a to b , there is no intermediate state other than a and b occurs;
- 3) The beginning and the end of an action complete instantaneously;
- 4) The beginning and the end of a desire complete instantaneously;
- 5) Sometimes, the end of an action and the change of a context value occur simultaneously if the context value change is directly triggered by the action. This kind of assumptions can be made according to the practical cases.

There are two kinds of context value determinisms in SiSL: **predicate context value determinism** and **functional context value determinism**.

The **predicate context value determinism** can be expressed as the following sentence:

$$p(o_1, \dots, o_n, s) \Leftrightarrow \Phi_p(o_1, \dots, o_n, s)$$

$\Phi_p(o_1, \dots, o_n, s)$ is a formula consists of predicates, connected by logical symbols. For example, a predicate context value determinism is:

$Loggedon(x, s) \Leftrightarrow$

$$\begin{aligned} & \{Loggedoff(x, preSit(s)) \wedge pageOf(preSit(s)) = \\ & (Page_ReviewerLogin \vee Page_AdminLogin) \wedge \\ & ActionIn(act_click(Btn_Login), preSit(s)) \wedge \\ & content(TextBox_Account, preSit(s)) = x \wedge \\ & content(TextBox_Password, preSit(s)) = pswOf(x)\} \vee \\ & \{Loggedon(x, preSit(s)) \wedge \neg ActionIn(act_click(Btn_Logoff), preSit(s))\}. \end{aligned}$$

The explanation of the above sentence is: the user x is logged on in situation s , if the following conditions are satisfied in the previous situation $preSit(s)$: the page is “*Reviewer_Login*” or “*Admin_Login*”, the user clicks button “*Submit*” and the content in the textbox “*account*” is x and the content in the textbox “*password*” is the password of x , while x is not logged on (all these conditions are observable); or x is logged on and the user doesn’t click the button “*log off*”.

A **functional context value determinism** is a sentence as:

$$f(o_1, \dots, o_n, s) = y \Leftrightarrow \Gamma(o_1, \dots, o_n, y, s)$$

$\Gamma(o_1, \dots, o_n, s)$ is a formula consists of predicates, connected by logical symbols. An example of functional context value determinism is:

$numOfPapers(s) = x \Leftrightarrow$

$$\begin{aligned} & numOfPapers(preSit(s)) = x-1 \wedge pageOf(preSit(s)) = Page_SubmitAbstract \wedge \\ & ActionIn(act_click(Btn_Submit), preSit(s)) \wedge error(preSit(s)) = NULL. \end{aligned}$$

The explanation of the first sentence is: the number of papers is x in situation s , if the following conditions are satisfied in the previous situation $preSit(s)$: the page is “*Submit Abstract*”, the user clicks button “*submit*” and there is no error on the page.

In some cases, the context values can only be obtained by several simultaneous actions. For example, character ‘Y’ can only be obtained by simultaneously pressing “*Shift*” and ‘y’ in one situation.

The benefits for defining context value determinisms are:

- 1) The context value determinisms are useful for inferring user’s actions when we cannot capture the actions directly but can only observe the context values;
- 2) The relations of some situations can be specified. In practice, we usually take a snapshot of a situation when action occurs or context value changes. Therefore, the relations of the consecutive situations in a situation sequence we capture in practice can be determined. Furthermore, the situations changing process in a sequence can be reasoned if significant action occurrences and context value changes can be captured.

(3) *Desire Satisfaction Axioms:* *A desire is satisfied when a set of context values are reached in the current situation.* Intuitively, a desire should be defined as a (set of) predicate among objects, and the desire is achieved when the (set of) predicate is true. However, a predicate cannot return a desire as the output (it only returns “true” or “false”), so desires should be defined as a sort of entities, and there is an additional desire satisfaction axiom for specifying the conditions under which a desire is achieved.

For every desire, there is a **desire satisfaction axiom** in the following form:

$$Satisfied(d(x_1, \dots, x_n), s) \Leftrightarrow H_d(x_1, \dots, x_n, s)$$

$H_d(x_1, \dots, x_n, s)$ is a formula consists of predicates, connected by logical symbols. For example:

$$Satisfied(submitBefore(x, t), s) \Leftrightarrow Submitted(x, s) \wedge gmt(s) < t$$

The translation of the above sentence is: the user's desire: submit x before time t , is satisfied in situation s if the status of x is "*submitted*" and the time-point of s is before t . The desire satisfaction axioms are used for determining satisfied situations in practice, because the context values in $H_d(x_1, \dots, x_n, s)$ should be observable.

According to Axiom I, actions performed can be regarded as an external reflection of human internal mental state in a specific circumstance. For example, an agent wants to log into his email inbox. When the network speed is good, he usually chooses the normal view; when the webpage stays on buffering for a long time, he may switch to a simplified view to speed up the process, or may keep on waiting. In both cases, the agent wants to satisfy the same desire, however, has two different possible actions, and his decision to choose which action to perform should conform to certain probability distribution depending on the context value (the network speed).

Axiom II provides a theoretical explanation of relations between actions and context values. It states that some context value changes are the results of agent's actions. These context value changes are directly or indirectly influenced by the actions, while other context values may not be influenced by any action, such as the time of the day. To build a complete domain knowledge base, the context values for all contexts, including those can be influenced by actions and those cannot, should be specified. While in practice, such context value determinisms can be encoded into a mathematical model through learning.

Axiom III indicates that desire changes may depend on the context values in current situation, because if the ideal context values are reached, i.e., the agent's desire is satisfied, a new desire is more likely to emerge, otherwise the desire is less likely to change.

Given the above axioms, we can establish certain determinative connections from desire to action, to context value, and then to desire changes. Furthermore, to completely describe the domain, the determinative relations from each desire to each action, from each action to each context value, and from each context value to each desire, all should be specified. However, in practice, agent's action selections and desire changes sometimes may be random and may not strictly conform to our axioms, because human are such complicated beings, and sometimes are unpredictable. But from a statistical point of view, based on long-time observations and with the help of suitable mathematical methods, we can try to build a computational model to reflect the patterns of agent's action selections and desire changes in those commonly seen situation sequences, which should roughly satisfy Axiom I, II and III, as I believe. Hence, the Axioms proposed above can be used as the criteria for selecting our observation means and mathematical methods.

CHAPTER 4. DESIRE INFERENCE AND NEW INTENTION DETECTION USING CRF

In this chapter, I present a methodology of new intention detection for software evolution. In the previous chapter, I explained the application fields of my methodology, which are human-centric context-aware domains. My methodology is supposed to be applicable in different types of software systems, e.g., dynamic websites, location-based mobile applications, smart home systems, etc., as long as they are human-centric and context-aware. I use the term “human-centric” to generalize the common nature of various application systems in which humans play a central role in driving system evolution, and the term context-aware to emphasize the physical properties of the system that is sensor-laden for monitoring users’ actions and system status. Different from many existing work [55], [56], [57] that are system/application-specific, my methodology targets at the human-centric character of a class of systems, making itself generally applicable, flexible, and adaptive in a wide range of applications. By characterizing and formalizing the nature of the target application domains, I aim at clarifying the following problems:

- To embody the human-centric characters, what essential entities and relations need to be formalized? And how to formalize them?
- To infer human desires, what kinds of states, i.e., data shall be captured? What’s the logical process of inference?
- How to choose suitable mathematical method to infer human desires based on observations?
- Based on observations and inferred desires, how to filter and formalize new intentions? And how to further elicit new requirements to enable software evolution?

To answer the above research questions, in the following paragraphs, I introduce the construction of domain knowledge base in section 4.1. Then in section 4.2, I introduce the principle of desire inference and new intention detection, and discuss the observation means, and the principle for desire inference and new intention detection. In section 4.3, I theoretically explain why the CRF method is chosen as the mathematical foundation, and propose a step-by-step methodology to infer human desires, followed by three methods for detecting users' new intentions. Finally in section 4.4, I discuss how to make corresponding improvement to the system based on detected new intentions.

4.1 Knowledge Base of a Human-centric Context-aware Domain

The domain knowledge base is the basis for desire inference, new intention detection and system evolution. The following diagram shows the basic elements in the SiSL domain knowledge base and its usefulness in our framework.

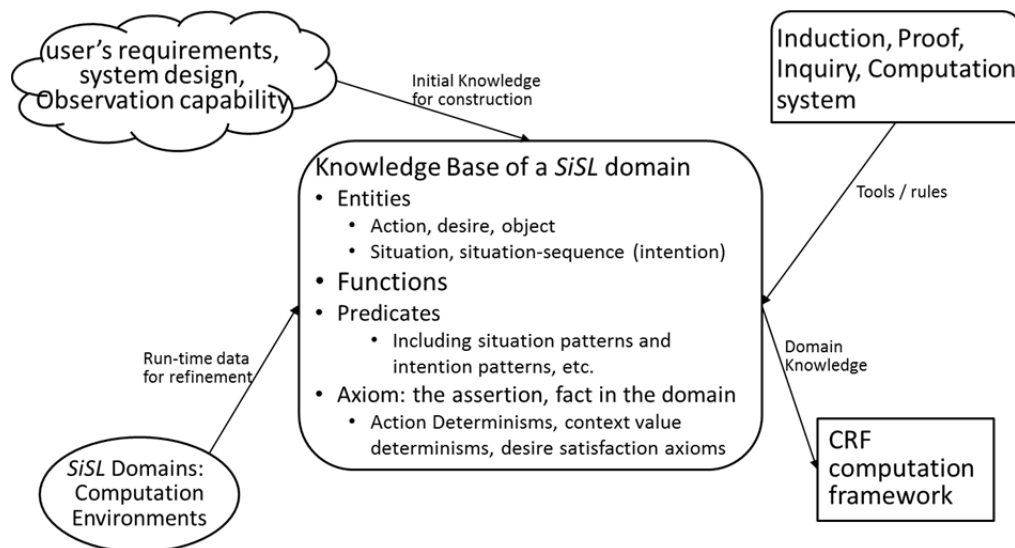


Figure 4.1 SiSL Domain Knowledge Base and its applications

In the SiSL domain knowledge base, there are the following items:

- (1) A set of different sorts of entities;

- Predefined entities: desires, actions and objects
 - Runtime entities situations, situation-sequences (including intentions);
- (2) A set of functions and predicates;
- Contexts, situation patterns and intention patterns;
 - Domain independent functions and predicates;
 - Domain specific functions and predicates.
- (3) A set of axioms;
- Action determinisms;
 - Context value determinisms;
 - Desire satisfaction axioms;

The prerequisite knowledge for constructing the initial SiSL domain knowledge base (*KB*) is user's requirements on the system: the use case scenarios and the *UML* models.

Knowledge Base Construction: the process of constructing the initial KB of a SiSL domain is:

- (1) Specify the use case scenarios based on user's requirements and the system design.

For example, a use case scenario "upload a paper" can be described as follows:

Active stakeholders: author

Process of uploading a paper:

- 1) Click the menu option "Upload a file";
- 2) Input paper ID and password, click button "Submit";
 - Alternative 2.1: click button "Send me my password";
 - Exception 2.1: wrong ID or password, error message shows on the page;
- 3) On the following page, click "choose file" to upload a file;

- Alternatives 3.1: modify contents in the form to change the paper information;
- 4) Click button “Submit”;
- Exception 4.1: error message shows on the page;

Outcome: The uploaded file is stored in the file folder; the item of this paper in the database is indicated as “uploaded”; a confirmation email is sent to the author; the confirmation message appears on the top of the page.

- (2) Define the *desires*, *actions*, *contexts* and *objects* based on the use case scenarios

Desires: a *desire* is defined for a use case scenario, which is usually a process for fulfilling a user’s goal. For example, a *desire: Des_UploadPaper* is defined for the use case scenario “upload a paper”;

Actions: user’s operations on the system involved in the use case scenario are defined as *actions*. The occurrences of *actions* should be observable by sensors deployed in the system. For example, the *actions: act_click(Menuopt_UploadPaper)*, *act_click(Btn_Submit)* can be defined to represent the operations: click menu option “Upload a file” and click button “submit”;

Objects: the physical and abstract objects involved in the use case scenarios are defined as *objects*. For example, *objects Menuopt_UploadPaper* and *Btn_Submit* are defined to represent menu option “Upload a file” and button “submit” respectively;

Contexts: the status of objects and the relations among objects involved in the use case scenarios are defined as contexts. Each context has a set of values, and the value set of a predicate context is {True, False}. The values of *contexts* should be observable by sensors deployed in the system. For example, *MsgOn(Error_NoFile, Page_UploadPaper, s)* is a predicate context and it represents that the error message *Error_NoFile* is (or not) on the page *Page_UploadPaper*.

- (3) Define situation patterns based on the actions, desires and contexts

Generally, a step in the use case scenario will be transferred to a situation pattern, which should contain the user's current desire, user's current actions and context values which are relevant for inferring user's desire.

For example, for the first step in the use case scenario "Submit a Paper", there is a corresponding situation pattern SP_1 defined as:

$$SP_1(s) \Leftrightarrow \\ DesireIn(Des_UploadPaper, s) \wedge ActionIn(act_click(Menuopt_SubmitPaper), s).$$

Some other *situation patterns* defined for the use case scenario "Submit a Paper" are:

$$SP_2(s) \Leftrightarrow DesireIn(Des_UploadPaper, s) \wedge ActionIn(act_click(Btn_Submit), s) \wedge \\ pageIn(s) = Page_PaperLogin;$$

$$SP_3(s) \Leftrightarrow DesireIn(Des_UploadPaper, s) \wedge ActionIn(act_click(Btn_SendPsw), s) \wedge \\ pageIn(s) = Page_PaperLogin;$$

$$SP_4(s) \Leftrightarrow DesireIn(Des_UploadPaper, s) \wedge ActionIn(act_click(Btn_ChooseFile), s) \wedge \\ pageIn(s) = Page_UploadPaper;$$

$$SP_5(s) \Leftrightarrow DesireIn(Des_UploadPaper, s) \wedge ActionIn(act_click(Btn_Submit), s) \wedge \\ pageIn(s) = Page_UploadPaper;$$

$$SP_6(s) \Leftrightarrow DesireIn(Des_UploadPaper, s) \wedge pageIn(s) = Page_UploadPaper \wedge \\ MsgIn(Msg_UploadSuccess, s).$$

The above situation patterns describe the possibly appearing situations in the process of uploading a file. However, these situation patterns haven't completely described the practical situations. For example, the possible error messages showing on the page, the contents in the input boxes are not included in any above six situation patterns.

- (4) Build the transition relationships for the predefined *situation patterns*

The transition relationships among different situation patterns should be specified so that intention patterns can be defined. For example, situations with situation pattern SP_1 may be followed by situations with situation pattern SP_2 or SP_3 , and situations with situation pattern SP_2 and SP_3 may be followed by situations with situation pattern SP_4 .

In first-order logic, the relations among predicates cannot be expressed, so we can draw a situation transition diagram to express the transition relationships of situation patterns. Each use case scenario has a situation transition diagram and the situation transition diagram should contain the normal steps, exceptions and alternatives in the use case scenarios. For example, the situation transition pattern of the use case scenario “Submit a Paper” is:

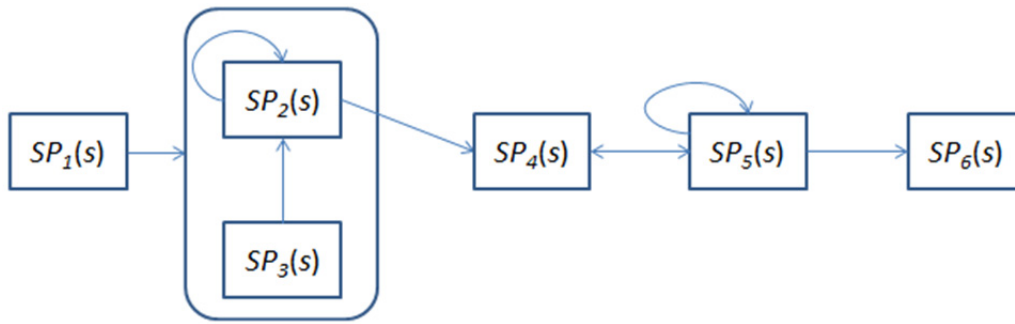


Figure 4.2 The situation transition diagram for uploading a file in MyReview

- (5) Define the intention patterns based on situation transition relationships

Generally, each use case scenario has a corresponding intention pattern, which is defined based on the situation transition relationships. For example, the intention pattern for the use case scenario “Submit a paper” is as follows:

$$\begin{aligned}
 IP_1(i) \Leftrightarrow & SP_1(\text{initialOf}(i)) \wedge SP_6(\text{finalOf}(i)) \wedge \{SP_2(\text{nextOf}(\text{initialOf}(i), i)) \vee \\
 & SP_3(\text{nextOf}(\text{initialOf}(i), i))\} \wedge \\
 & \{\forall s.SP_2(s) \wedge \text{SituationIn}(s, i) \Rightarrow SP_2(\text{prevOf}(s, i)) \vee SP_3(\text{prevOf}(s, i))\} \wedge \\
 & \{\forall s.SP_4(s) \wedge \text{SituationIn}(s, i) \Rightarrow SP_2(\text{prevOf}(s, i)) \vee SP_5(\text{prevOf}(s, i))\} \wedge \\
 & \{\forall s.SP_5(s) \wedge \text{SituationIn}(s, i) \Rightarrow SP_4(\text{prevOf}(s, i)) \vee SP_5(\text{prevOf}(s, i))\} \wedge
 \end{aligned}$$

$$SP_3(\text{prevOf}(\text{final}(i), i)).$$

As shown in the above example, an intention pattern is defined as a predicate and specified using an axiom. After defining intention patterns for all the use case scenarios, the initial domain knowledge base is constructed, which contains predefined desires, actions, objects, contexts, situation patterns, intention patterns and other predicates and functions.

4.2 Desire Inference and New Intention Detection

The SiSL domain knowledge base introduced above will be used as the basis of desire inference and new intention detection, which will be applied for driving software evolutions. A proper mathematical method, e.g., CRF, should be designed to enable the computation of optimal results in the process of desire inference and new intention detection.

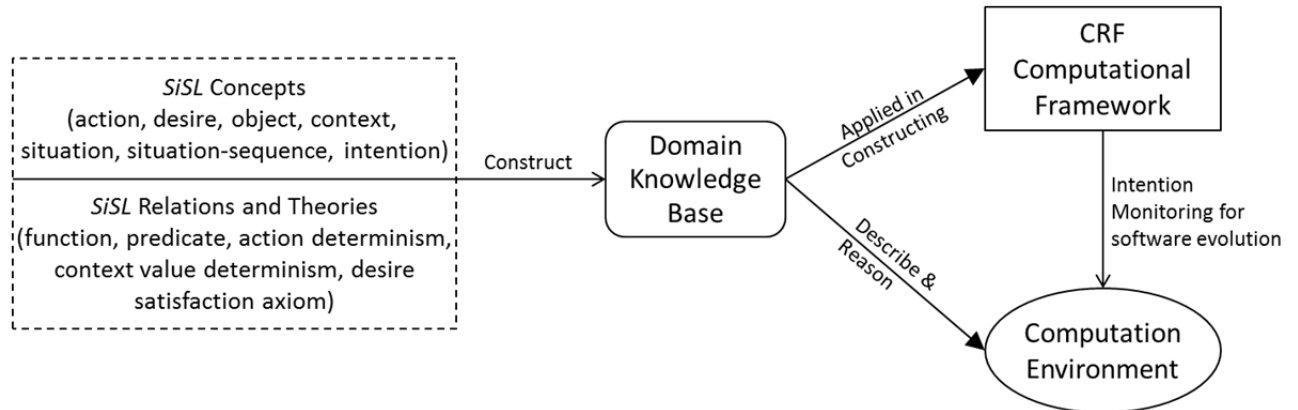


Figure 4.3 Intention Monitoring Based on SiSL-Domain Knowledge Base

As the precondition for new intention detection, desire inference is to infer agent's desire at each observation time point. According to the axioms introduced in the previous chapter, it should be possible to infer agent's desires based on his/her actions and current context values. Intuitively, HMM is a feasible method because it is used to predict hidden states (desire) from observable variables (actions and context values), and it is also considered capable of determining human mental states from human actions [20]. In fact, I initially started with HMM and

eventually chose a similar method, CRF, which is now considered as a better fit for our methodology. More detailed comparison between these two methods will be introduced in a later section 4.3. But before delving into the depth of mathematical model selection, it is necessary to design the data first.

To accurately infer agent's desires, I propose to observe the following states within the observation capability:

- 1) Agent's actions within the system domain, especially operations on the system interface;
- 2) Those contexts whose values may influence agent's actions;
- 3) Those contexts whose values can be changed by agent's actions;
- 4) Those contexts whose values can indicate whether agent's desire is satisfied or not;
- 5) Other context values which may not be considered as of any significance, but can be easily captured. The motivation to include these data is to construct raw data for future analytics, since the data may become valuable and relevant at some point.

After determining the contents to observe, I further suggest the instants at which an observation should be made:

- 1) Action trigger: each time when the agent performs an action;
- 2) Context-value change trigger: each time when there is a change of the context value being monitored;

The above principles to determine observation contents and observation frequency are given as general guidelines, as each individual system may have its own characteristics and limitations, and shall be carefully handled in a case-by-case basis. However, even for the same system/domain, different observation setup may capture different sets of raw data, which may

lead to different inference results. In order to get most out of our methodology, it is recommended to experiment with different data collection mechanisms and methods, and select the one with the best performance.

After setting up the monitoring mechanism, now I introduce the principle of desire inference and new intention detection, as well as related concepts as follows (A step-by-step methodology is given in section 4.3):

1) Observation: An observation is the observed status of the system domain at a time point, including a set of actions and context values that are time-stamped. Desire inference and new intention detection will be performed based on observations. Based on the observation contents and frequency discussed above, a number of applicable and necessary sensors should be deployed in the system to capture agent's actions and relevant context values. Some actions and context values cannot or will not be captured due to observation capability or non-necessity, hence an observation may only contain only a part of the status of the domain.

2) Desire Inference: In desire inference, the inference model takes a sequence of observations as the input and outputs the agent's desire at each observation time point. As I mentioned at the beginning of this chapter, a mathematical inference model is used in this process. Feasible candidates can be HMM, CRF, etc.

3) Intention Inference: After the agent's desire is inferred for each observation, the intention for achieving a certain desire can be obtained by connecting the situations with the same desire into a sequence that is destined to reach a goal state. The underlying rationale is because of our definition of intention in section 3.4.

4) New intention detection: A new intention is a situation-sequence pattern which has not been predefined in the domain knowledge base, i.e., an agent's new behavior sequence pattern

for achieving a desire. I believe that theoretically a new intention will arise in two cases: either the agent has a new desire, or the agent has a new strategy for achieving a predefined desire. In practice, new intentions will also be detected when the agent cannot properly perform operations due to system flaws, which may cause their divergent behaviors. All of these cases are useful for system evolution, because they can imply agent's new requirements or system drawbacks. Therefore, the ultimate goal of our methodology can be reached, that is to detect agent's new intentions for driving system evolution.

4.3 Using the CRF Method for Desire Inference and New Intention Detection

The Conditional Random Fields (CRF) method is applied in our research to build the computational model for desire inference and new intention detection. More specifically, I use linear-chain CRF (refer to section 2.3). Compared with HMM, linear-chain CRF has more advantages on labeling sequential data and is supposed to be more suitable for our research due to the following theoretical reasons:

1) According to Axiom I (see section 3.6), the agent's actions are usually determined by their current desire and current context values. Such determinative relation can be reflected in a CRF model through defining feature functions that encode the known relations between the attributes of the current observation (most likely context values) and the current desire. However, in HMM, Axiom I cannot be well reflected because the current action is only determined by the current desire according to the state-observation emission relations [20].

2) According to Axiom II (see section 3.6), some context value in a later observation may depend on context values and an agent's actions in a former observation. In CRF, such relations can be encoded in some feature functions to better infer the current desire. However, in

HMM, the relations between consecutive observations cannot be reflected because the observations are supposed to be independent (output independent assumption [20]);

3) According to Axiom III (see section 3.6), the satisfaction of a desire depends on current context values. As an agent usually moves on to fulfill a new desire when their current desire is just satisfied, desire transitions also depend on current context values. Such desire satisfaction determinism and desire transition principle can also be reflected in a CRF model using feature functions that encode the known relations among the attributes of the previous observation, the previous desire and the current desire. In HMM, according to the Stationarity Assumption [20], the desire transition probabilities are stationary regardless of the current context values, so the desire transition principle cannot be well represented.

Table 4.1 Comparison Between HMM and CRF for Desire Inference

	HMM	Linear-CRF
Axiom I	Not supported because of state-observation emission relations	Supported by defining feature functions to reflect Action Determinisms
Axiom II	Not supported because of Output Independent Assumption	Supported by defining feature functions to reflect relations between consecutive observations
Axiom III	Not supported because of Stationarity Assumption	Supported by defining feature functions to reflect Desire Transition based on context values
Assumption 2	Supported because of Markov Property	Supported because of linear property

Based on the above reasons (also shown in Table 4.1), linear-CRF is chosen as the mathematical foundation for our methodology of desire inference and new intention detection.

The general format of feature functions in the linear-chain CRF model in this thesis is formulated

as $f_n(O, i, d_i, d_{i-1})$, in which:

- O is a sequence of observations;
- i is the ordinal number of current observation in O ;
- d_i is the inferred desire for the i th observation in O ;
- d_{i-1} is the inferred desire for the $(i-1)$ th observation in O ;
- The output is 1 when certain relations specified by the function are satisfied among O , d_i and d_{i-1} , otherwise the output is 0 .

The fundamental task of desire inference using CRF is to find the desire sequence D^* with the largest labeling score (highest probability), and use D^* as the inference result for the input observation sequence. Technically this task is very similar to the one that formula (2) (in section 2.3) aims at, which is to predict word labels for a sentence. So I can modify formula (2) by replacing its parameters with our own, and have formula (3) as the core mathematical model for desire inference for observation sequence $O = \langle o_1, \dots, o_n \rangle$:

$$D^* = \langle d_1^*, \dots, d_n^* \rangle = \operatorname{argmax}_D \left(\frac{\exp\left[\sum_{j=1}^m \sum_{i=1}^n \lambda_j f_j(O, i, d_i, d_{i-1})\right]}{\sum_{D'} \exp\left[\sum_{j=1}^m \sum_{i=1}^n \lambda_j f_j(O, i, d'_i, d'_{i-1})\right]} \right) \quad (3)$$

- λ_n is the weight associated with f_n .

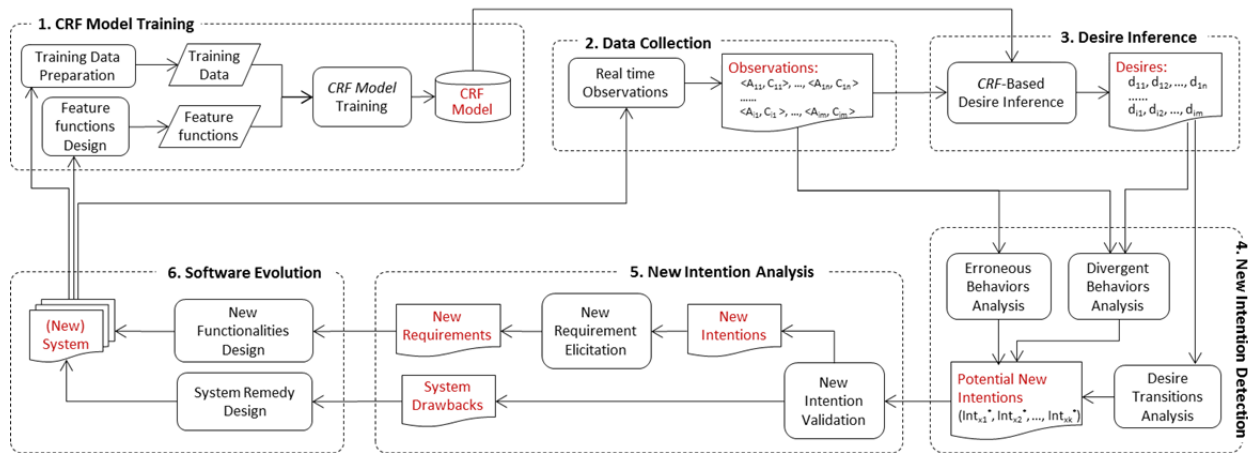


Figure 4.4 Methodology for Desire Inference and New Intention Detection.

The complete process of applying our methodology to detect new intentions for software evolution is depicted in Figure 4.4.

The detailed description of each step's task is introduced as below:

Step 1: Constructing the CRF model. To achieve the two goals in our study: (1) accurately infer the agent's desires predefined in the domain knowledge base; (2) detect the agent's new or unexpected intentions, a linear-chain CRF model that encodes users' known behavior patterns for achieving desires should be built as the metrics for outlier detection [31]. I propose to use supervised learning [51] to train the CRF model.

Step 1.1: Training Data Collection: The input training data consists of a set of sequences of observation, and each of the sequence shall follow the form of $\langle o_1^*, d_1^* \rangle, \langle o_2^*, d_2^* \rangle, \dots, \langle o_m^*, d_m^* \rangle$, in which each observation o_i^* is labeled with a desire d_i^* . Since the CRF model is used as a standard reference model to infer desires, it must be built upon the existing domain knowledge, including known or predefined desires, behavior patterns, etc. To make it happen, the observation sequence $\langle o_1^*, o_2^*, \dots, o_m^* \rangle$ in our training data shall be collected from observing user behaviors that are expected to conform to the system design, e.g., existing use case scenarios and sequential diagrams, etc., and the desires associated with each observation $\langle d_1^*, d_2^*, \dots, d_m^* \rangle$ shall be accurately reported by users or manually labeled by domain experts.

Step 1.2: Defining Feature Functions: Based on domain experts' knowledge, feature functions can be defined to reflect the relations between observations and desires. If using existing tools, such as CRF++ [52], Flex-CRF [53], etc., we need to design feature templates which specify the form of feature functions (will be introduced in section 5.4 with examples.)

Step 1.3: Training: With the training data prepared in Step 1.1, and the feature functions designed in Step 1.2, we can start training the CRF model. Existing tools are available for

building the CRF model, each with its own technical merits. Alternatively, people can design and write their own algorithm based on their own preference.

The main task in this step is to get the CRF model constructed and ready for upcoming inference work. To get the best inference result, it is recommended to iterate the steps from 1.1 to 1.3, and to use different training data sets with different feature functions configured for training, so that a relatively-optimal model can be acquired.

Step 2: Data Collection and Pre-processing: Observe agent (real user)'s actions and relevant context values when they operate on the system, and generate observation sequences $\langle o_{11}, \dots, o_{1n} \rangle, \langle o_{11}, \dots, o_{1m} \rangle, \dots;$

Step 3: Desire Inference: Input the observation sequences prepared in Step 2 into the CRF model acquired in Step 1, and infer the agent's desire at each observation time point.

Step 4: New Intention Detection: New Intention Detection: As introduced in section 4.2, a new intention is a situation-sequence pattern which has not been predefined in the domain knowledge base. Considering the various causes of new intentions, and for the purpose of system improvement, I propose three methods for detecting new intentions based on the results of desire inference:

- **Method I: Divergent behavior analysis.** Detect users' divergent behaviors through desires inferred with low confidence (probability). Divergent behaviors usually occur when a user doesn't follow an expected way to operate the system (a predefined intention), which may indicate their dissatisfaction on the system or new desires. For example, on a ticketing system, a user might think the process is too tedious so he tried different ways to skip some steps or started over to find if there is any speedy entry. His behaviors may appear irregular and cannot be well interpreted by the CRF model, which will result in low confidence when

labeling desires. Therefore, these divergent behaviors can be singled out for analyzing users' new requirements.

- **Method II: Desire transition analysis.** Obtain a new intention based on desire transitions. If two desires often appear consecutively, a new intention can be considered to make the desire transitions more smoothly and efficiently. Accordingly, the system can be modified to simplify users' operations. For example, again on the ticketing system, if users often move on to select a seat right after finishing buying a ticket, a link that directs the users to the seat selection page can be added on the confirmation page of "buying a ticket". This kind of desire transitions can be obtained based on the results of desire inference with high confidence level.

- **Method III: Erroneous behavior analysis.** Find new desires and system drawbacks from users' erroneous behaviors. When users have new desires which are not supported by the current system, their behaviors may trigger error reports. For example, on the same ticketing system, if a user tries to input a phone number in a format that is not supported by the system, an error report will be generated. If such error report occurs time and time again, it may indicate many users actually want to input their phone number in other non-supported format, which can become a new intention. Additionally, some of the detected users' erroneous behaviors can be false positive, which means these behaviors are actually normal, while some system drawbacks/defects making them look "erroneous". So these behaviors with related error reports can be a good starting point to detect system drawbacks and defects, and to further improve the system.

4.4 System Evolution Process Based on New Intention Detection

The detected potentially new intentions and system drawbacks will be further analyzed

by domain experts to determine whether they are indeed new intentions or system drawbacks. This step cannot be done automatically, and must be manually performed by domain experts. The system drawbacks may not be too difficult to identify, while the new intentions are usually implicit and hard to decide. Domain experts can make some assumptions of the new intentions and then verify them based on their expertise, or they can directly inquire the users for evaluation if possible. After this validation and verification process is done, the current system shall be redesigned to satisfy users' new intentions, or be corrected to overcome those revealed drawbacks/defects. The complete process of system evolution based on new intention detection using the CRF method is shown in Figure 4.5.

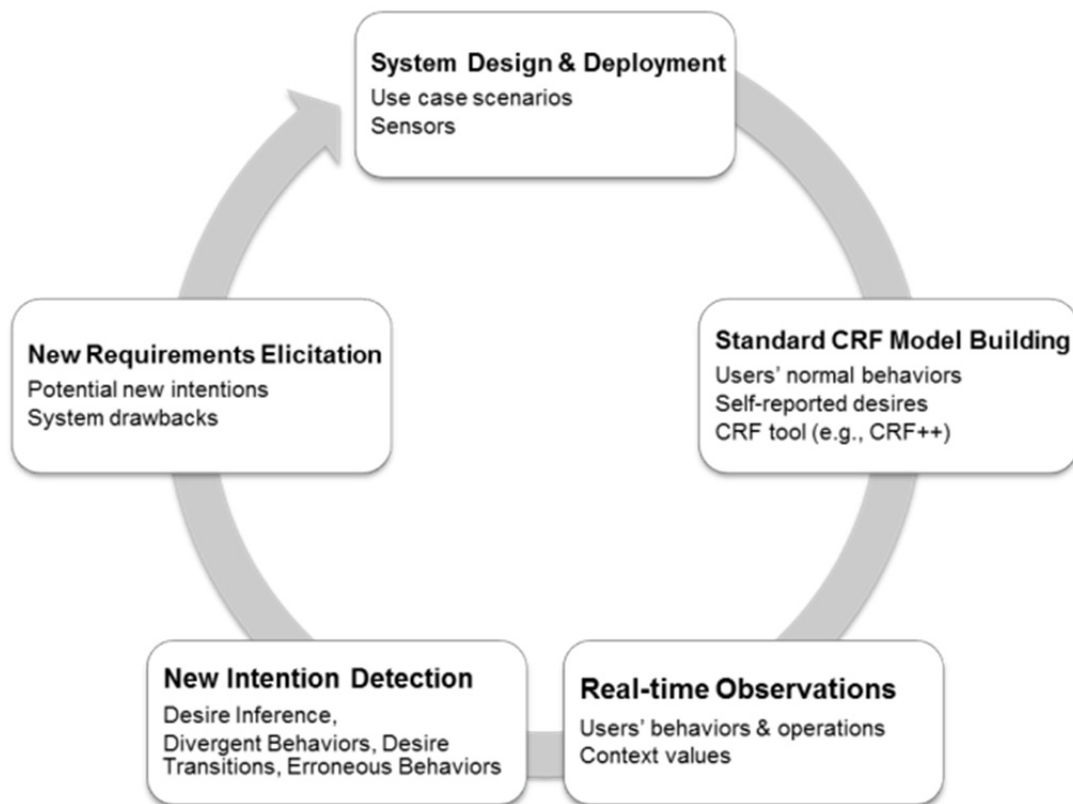


Figure 4.5 System evolution driven by new intention detection using CRF.

In summary, our methodology is supposed to be more efficient and effective for system evolution than traditional approaches due to:

1) Our methodology is based on observations of users' real-time behaviors, while traditional approaches often depend on users' delayed feedbacks, system defect reports or system performance logs, etc. [15], [13], [54]. Therefore, our methodology can shorten the time to get useful information for analyzing new requirements;

2) Observations of actions and relevant context values contain richer and more meaningful information for new requirements elicitation because they are the direct reflection of users' internal mental states that occur during their operation on the system. On the contrary, users' feedback is usually not as accurate yet implicit, while other resources such as system defect reports and system performance logs cannot reflect users' new requirements on the system effectively;

3) The application of the mathematical method linear-chain CRF enables accurate and rapid inference of users' desires, making our methodology more efficient than traditional manual analysis. The three methods proposed in section 4.3 for new intention detection can be automated as well. Hence, they can obtain useful information quickly and speed up the process of new requirements elicitation;

4) The new requirements discovered in our methodology are directly elicited from user's unexpected behaviors in the operational environment, thus the system can be pertinently improved to adapt to those user behaviors. Therefore, the system evolution path is supposed to be more appropriate and effective.

CHAPTER 5. EXPERIMENT ON A RESEARCH LIBRARY SYSTEM

As stated in the previous chapter, I believe our methodology is beneficial for efficiently and effectively discovering new requirements and shortening the software evolution cycle. In this chapter, I present an exploratory experiment on a dynamic web-based system to demonstrate and validate our methodology. First of all, I made the following hypotheses before conducting the experiment:

(1) Based on observations of user's actions and relevant context values, it is possible to accurately infer user's desires using a CRF model. I expect the inference accuracy will be 90% or higher and it should be higher than that by using HMM;

(2) It is possible to detect some new intentions based on the results of desire inference using the three methods introduced in section 4.3;

(3) New user requirements or system drawbacks can be revealed based on the detected new intentions; Rapid system evolution can be achieved based on the revealed new user requirements or system drawbacks.

To validate the above hypotheses, our experiment can be divided into two sub-experiments and one case study as follows:

(1) First-round experiment: An experiment to validate that high desire inference accuracy can be achieved by using CRF.

(2) New-intention-detection case study: A case study to demonstrate our methodology for new intention detection and new requirements elicitation;

(3) Second-round experiment: An experiment to validate the system improvement between two versions of systems.

5.1 Experiment Platform – the CoRE System

The experimental system is an online library system, called Cooperative Research Environment (CoRE). It was modified based on an open-source web application, MyReview [55], for managing the process of paper submission and paper review. The original system has served many academic conferences, and our modification still keeps its basic functionalities. Therefore, our experiment is expected to emulate users' operations on a real-world system. Another reason for choosing a web-based system to conduct our experiment is that it is easier to get participants' self-reported desires which are used to validate our methodology.

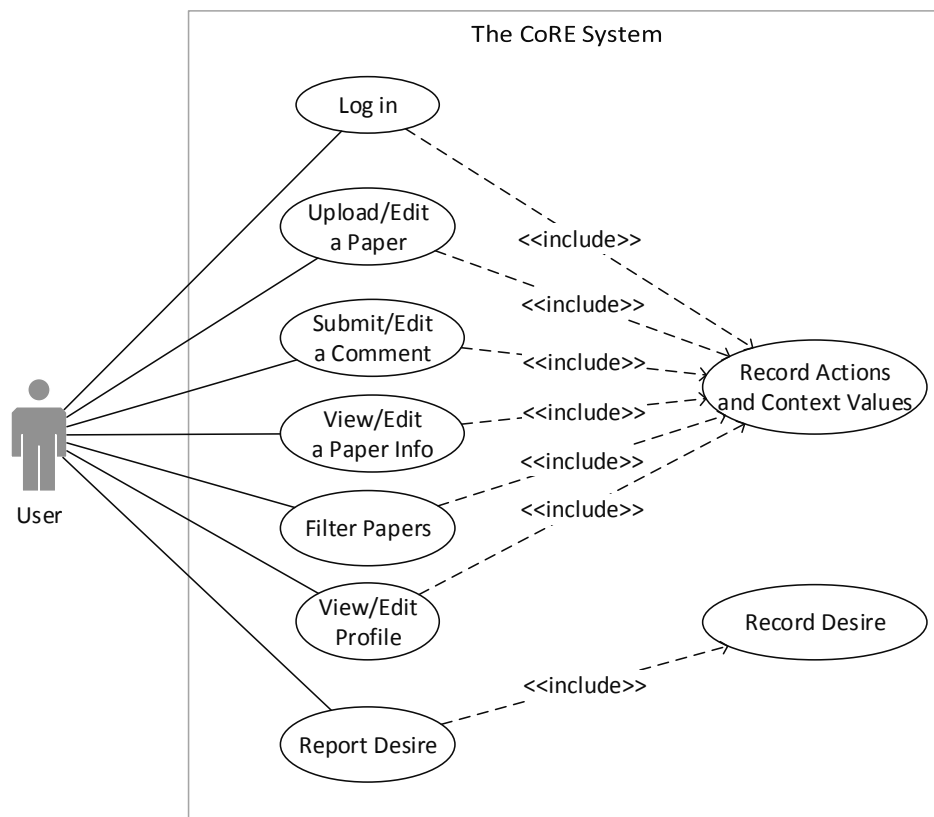


Figure 5.1 The use case diagram of the CoRE system.


CoRE has been designed and developed as a research community for people to share their thoughts and views on academic papers. Users can upload research papers, submit comments for papers, and view papers' information, etc. Figure 5.1 is the use case diagram of CoRE.

Home My account Member Admin

Please upload the paper using the form below. Note that all the fields are mandatory.

Paper upload form

Title	<input style="width: 95%;" type="text"/>		
List of authors	First name	Last name	Affiliation
	<input style="width: 95%;" type="text"/>	<input style="width: 95%;" type="text"/>	<input style="width: 95%;" type="text"/>
	<input style="width: 95%;" type="text"/>	<input style="width: 95%;" type="text"/>	<input style="width: 95%;" type="text"/>
	<input style="width: 95%;" type="text"/>	<input style="width: 95%;" type="text"/>	<input style="width: 95%;" type="text"/>
	<input style="width: 95%;" type="text"/>	<input style="width: 95%;" type="text"/>	<input style="width: 95%;" type="text"/>
	<input style="width: 95%;" type="text"/>	<input style="width: 95%;" type="text"/>	<input style="width: 95%;" type="text"/>
	<input style="width: 95%;" type="text"/>	<input style="width: 95%;" type="text"/>	<input style="width: 95%;" type="text"/>
	<input style="width: 95%;" type="text"/>	<input style="width: 95%;" type="text"/>	<input style="width: 95%;" type="text"/>
Other authors	<input style="width: 95%;" type="text"/>		
Key words	<input style="width: 95%;" type="text"/>		
Publisher	<input style="width: 95%;" type="text"/>		
DOI Digital Object Identifier	<input style="width: 95%;" type="text"/>		
Publication type	Conference ▾		
Publish date	Jan. ▾	2015 ▾	
Paper type	Practice ▾		
Upload	No file chosen		<input type="button" value="Upload File"/>
Paper format	<input checked="" type="radio"/> PDF		



Software Engineering Research Group
Department of Computer Science
Iowa State University
<http://ficse.cs.iastate.edu>

Desire:
Upload a Paper ▾

Drop-down menu for participants to report their desire

Figure 5.2 Interface of the page for uploading a paper in CoRE Version I.

To monitor users' behaviors and capture related context values, an embedded program is deployed in CoRE as a sensor. Users' operations, paper and comment submissions, and the contents on the web pages will be recorded and stored in the database. During each experiment session, users need to report/select their current desires from a dropdown list containing a set of expected desires. Examples of desire options are "Upload a paper", "Submit a comment", "View a paper information", and "Not in the list", etc. The users can also correct their desire selections in the post-session questionnaire in case that they forgot to report desires or reported an incorrect

one during the experiment. Figure 5.2 shows the interface of the page for uploading a paper in the CoRE system, on which we can see there is dropdown menu in the right-side pane for users to report their desires during the experiment.

5.2 Procedure of an IRB Approved Experiment

Due to the nature of our experiment involving human subjects, and to stay complied with federal regulations set forth by the Department of Health and Human Services and the Food and Drug Administration, all of the principle investigators in our experiment have completed the National Institutes of Health (NIH) Web-based training course “Protecting Human Research Participants”, and they have also closely worked with our local Institutional Review Board (IRB) for approval for conducting our experiment, which has been granted before our experiment was implemented.

More than 120 people participated in our experiment. Each participant was required to study the user manual, and had a chance to do some test operations on the system to get a preliminary understanding about it. Participants’ actions, self-reported desires, and relevant context values are recorded as the experiment raw data. In order to show our methodology’s ability to enable and speed up the evolution process of the CoRE system, the experiment has been done for two rounds, which is to emulate one software evolution cycle. The whole experiment procedure is described as below:

- 1) Deploy the initial system: CoRE Version I;
- 2) Run experiment round 1 for 30 days: invite participants, collect data records of participants’ actions, desires and context values;

- 3) Shutdown CoRE Version I. Apply our proposed methodology on the raw data captured in round 1, analyze and elicit users' new requirements. Revise the system accordingly;
- 4) Deploy the enhanced system: CoRE Version II;
- 5) Run experiment round 2 for 30 days: invite new participants, collect data records of participants' actions, desires and context values;
- 6) Shutdown CoRE Version II. Apply our proposed methodology on the raw data captured in round 2, analyze and elicit users' new requirements.
- 7) Evaluate the effectiveness of our methodology on the evolution of CoRE from Version I to II.

During the experiment in each round, participants can enter CoRE for multiple times, and each time is recorded as one session. In each session, participants shall follow the procedure as follows:

- 1) Visit experiment website and log into experiment;
- 2) Answer pre-session questionnaire about their familiarity with the system;
- 3) Start a session: Log into CoRE and start operating on the system. Participants are free to carry out any operations, but need to report their desires on each webpage through choosing a predefined desire in a dropdown list.
- 4) End a session: Participants can end a session at any time they prefer. Then they will be directed to a post-session questionnaire, where they can give some feedback of the system, and also have a chance to correct their reported desires if necessary.

Observation in our experiment is action triggered. During each experiment session, the embedded monitoring program in the system will take a snapshot of the participant's action and some system context information when he/she performs an operation on the system.

Each raw data record has the following attributes, with an example shown in Table 5.1:

- 1) Time: the time point when the participant performs an operation;
- 2) Participant's login ID;
- 3) Action: including mouse click on a button or a link, or selection on a dropdown menu;
- 4) The current webpage where the action occurs;
- 5) Contents on the webpage (user's submitted input, system's responses to the user's action including exceptions and error messages);
- 6) Participant's self-reported desire.

Table 5.1 Example Raw Data Record

Record Item	Data
Time	2014-06-23 12:11:20
loginID	User020
Action	click(Btn_Login)
Page	Page_Login
Content	[Login ID]Test001 [Password]112233 [Message]Invalid password
Desire	Filter Papers

5.3 Data Collection and Preprocessing

This step corresponds to Step 2 in our methodology (in section 4.3). In our experiment, Step 2 and Step 1 in our methodology have been done reversely, because several domain experts and system designers were also participants, and their own data records were mixed with others and were later filtered out as training data for building the CRF model. For other application, if

there is a good historical data source available that is suitable for training purposes, one can certainly follow all steps of our methodology in the normal order.

There are 10,063 raw data records and 585 experiment sessions captured in the first round of experiment, and 10,524 raw data records and 582 experiment sessions captured in the second round. A raw data record contains all the attributes shown in Table 5.1. A record of an experiment session is the data sequence that starts when the user logs into the experiment, and ends when the user logs out.

As the accuracy of participants' self-reported desires is critical for validating the results of desire inference, those raw data records with inaccurate self-reported desires are not usable and are considered as noise. To remove those noise, raw data records have to be checked and filtered, noisy data in a unit of session will be removed based on the following principle:

(1) If most (>50%) of self-reported desires are "Not in the list", which is the default value in the desire selection dropdown list. I will assume that in this case the participant forgot to report his desire at all or most of the time in the experiment;

(2) If the answer is "no" for the question "Did you select the desire every time when you had a new desire?" in the post-session questionnaire;

(3) After filtering based on (1) and (2) is done, I had the rest of the data records manually checked by domain experts and system designers, and evaluated in the perspective of whether the participants' self-reported desires were reasonable or not. Those obviously wrong ones should and have been removed because they cannot be used to validate our desire inference result. A case of example is that a participant might forget to report his desire (change) when he accomplished his previous task and started a new one.

After preprocessing and noise filtering, the final data set of the first-round experiment has 6880 data records and 369 experiment sessions, and the final data set of the second-round experiment has 6931 data records and 361 experiment sessions.

5.4 Building the Standard CRF Model

According to Step 1 in section 4.3, a CRF model shall be built and used as measure metrics to infer users' desires and detect new intentions. The training data records for building the CRF model shall be collected from the system designers' and experienced users' behaviors. As I mentioned in 5.3, in our experiment, the training data records are actually mixed with others. However, they have the following attributes and can be readily filtered out:

- 1) The records belong to system designers and participants whose behaviors were conducted to achieve certain expected desires. By "expected", it means those behaviors shall conform to the CoRE use case scenarios, including normal cases, exceptions, and alternatives;
- 2) The training data set shall cover all of the expected desires;
- 3) The self-reported desires in records in the training data set shall be accurate.

After being processed from the raw data, a training data set was acquired for building the CRF model. There are 2930 data records and 158 experiment sessions in the training data set of the first-round experiment. A sample of these training data is given in Table 5.2.

Table 5.2 Sample Training Data

Observation	Time Interval	Desire
clickMenuAllPapers	30s	ViewAllPapers
clickPaperInfos&PaperID	m	ViewAPaperInfo
clickFilter&FilterCategory	60s	FilterPapers

Each training data record has three elements: observation, time interval and desire.

- Observation: includes the action and context values which are captured simultaneously. The format is *action&contextvalues*, e.g., in the observation *clickFilter&FilterCategory*, *clickFilter* is an action and *FilterCategory* is a context value;
- Time interval: the time interval between two consecutive observations. A time interval is calculated based on the time point in each data record. The partition of different time intervals is given in Table 5.3.

Table 5.3 Partition of Time Intervals

Actual duration of the interval	Time Interval
0 ~ 5s	5s
5s ~ 10s	10s
10s ~ 20s	20s
20s ~ 30s	30s
30s ~ 60s	60s
60s ~ 60mins	m
>= 1 hour	h

- Desire: user's self-reported desire.

I use an open-source tool called CRF++ [52] to build our CRF model based on the training data. According to Step 1.2 in 4.3, we need to design the feature functions for our CRF model, and in CRF++, they can be automatically generated based on feature templates which specify their formats. The feature templates used in our experiment are shown in Table 5.4, in which each entry denotes one template.

Table 5.4 Feature Templates

Unigram feature templates	Bigram feature templates
U01:%x[0,0]	B01:%x[0,1]
U02:%x[-1,0]/%x[0,0]	B02:%x[0,0]/%x[0,1]
U03:%x[0,0]/%x[1,0]	B03:%x[-1,0]/%x[0,1]
U04:%x[-2,0]/%x[-1,0]/%x[0,0]	B04:%x[-1,0]/%x[0,0]/%x[0,1]
U05:%x[-1,0]/%x[0,0]/%x[1,0]	
U06:%x[0,0]/%x[1,0]/%x[2,0]	

In each template, special macros in the format of %x[*row,col*] will be used to specify a token in the input data. *row* specifies the relative position from the current focusing token and *col* specifies the absolute position of the column. The above feature templates were acquired after experimenting with the data for the highest inference accuracy. Below are some brief explanations of some key feature templates and why they are good for building the CRF model:

1) *U01~U06*: the unigram templates specify the relation between the observations and the current desire. %x[-1,0], %x[0,0] and %x[1,0] represent the previous, the current and the next observation respectively. An example feature function generated by *U02*: %x[-1, 0]/%x[0, 0] based on data records in Table 5.2 is:

$$f_1(O, n, d_{n-1}, d_n) = \begin{cases} 1 & \text{if } d_n = \text{ViewAPaperInfo} \\ & O_n = \text{clickPaperInfos\&PaperID} \\ & O_{n-1} = \text{clickMenuAllPapers} \\ 0 & \text{Otherwise} \end{cases}$$

The above feature function can be interpreted as: if the current observation O_n is click the link “*View Paper Infos*” of a paper and its previous observation O_{n-1} is click the menu option “*All*

Papers”, the current desire d_n is supposed to be *View a paper’s information* with a labeling confidence w_l associated with f_l .

Because of the nature of the domain of CoRE, it can make the inference results more accurate by considering the neighboring observations when designing feature templates. For example, there is a segment in the training data set shown in Table 5.5:

Table 5.5 Example Segment in the Training Data Set

Observation	Time Interval	Desire
clickLogin&LoginGood	30s	EditProfile
clickMenuMyProfile	10s	EditProfile
clickSubmit&ProfileUpdated	30s	EditProfile

The observation *clickLogin&LoginGood* is labeled as *EditProfile* because the following observations show that the user updates his profile. Another segment (Table 5.6) is:

Table 5.6 Example Segment in the Training Data Set

Observation	Time Interval	Desire
clickLogin&LoginGood	10s	ViewProfile
clickMenuMyProfile	10s	ViewProfile
clickMenuAllPapers	30s	ViewAllPapers

Here the observation *clickLogin&LoginGood* is labeled as *ViewProfile* because the following observations show that the user didn’t update his profile so he probably just viewed the profile. This kind of relations between observations and desires shown in the above two examples can be encoded into the CRF model based on template $U06:\%x[0,0]/\%x[1,0]/\%x[2,0]$, which takes the current, next and after next observations into account for inferring the current

desire.

2) *B01~B04*: the major difference between unigram templates and bigram templates is that the latter ones consider the previous desire for inferring the current desire but the former ones do not. These four bigram templates specify the relations among observations, time intervals, current desire and previous desire. An example feature function generated by *B02*: $\%x[0, 0]/\%x[0, 1]$ based on data records in Table 5.2 is:

$$f_2(O, n, d_{n-1}, d_n) = \begin{cases} 1 & \text{if } d_n = \text{ViewAPaperInfo} \\ & d_{n-1} = \text{ViewAllPapers} \\ & O_n = \text{clickPaperInfos\&PaperID} \\ & \text{TimeInterval}(O_n) = m \\ 0 & \text{Otherwise} \end{cases}$$

The above feature function can be interpreted as: if the current observation O_n is click the link *View Paper Infos* of a paper and the previous desire d_{n-1} is *View all papers*, the current desire is supposed to be *View a paper's information* which is not consistent with the previous desire because the time interval between two observations is m (which is long).

Because of the nature of the domain of CoRE, it can make the inference results more accurate by adding the factor of time intervals into feature templates. Table 5.7 shows two example segments in the training data set (separated by dotted lines).

Table 5.7 Example Segment in the Training Data Set

Observation	Time Interval	Desire
clickMenuUploadPaper	10s	UploadPaper
clickSubmit\&NoFile	m	UploadPaper
.....
clickMenuUploadPaper	10s	Test
clickSubmit\&NoFile	30s	Test

The first segment in Table 5.7 shows that if the user's desire is to *Upload a paper*, the time he stays on the page *Upload Paper* will be long (more than 1 minute), and if the user just wants to do some Test, the time interval is most likely short. This kind of relations among observations, time intervals and desires can be encoded in the feature functions generated by template B02, which takes the current observation and current time interval into account for inferring the current desire.

For the similar reason, time gaps between observations are important to desire inference as careless or unintentional errors are usually corrected shortly after they are made. An example of this case is shown in Table 5.8:

Table 5.8 Example Segment in the Training Data Set

Observation	Time Interval	Desire
clickMenuMyPapers	10s	UploadPaper
clickMenuUploadPaper	5s	UploadPaper
clickSubmit&PaperInfoGood	m	UploadPaper

The above example shows that the participant's desire was to *Upload a paper*, but he clicked link *My uploaded papers* first by mistake and then clicked link *Upload a paper* in 5 seconds. The accidental erroneous operation will be still labeled correctly if considering the short time interval between the current and the next operation. This kind of relations can be represented in template B01, which encodes the relations among the current time interval, previous desire and current desire. With the template B01 and data records in Table 5.8, a feature function will be generated which labels consecutive observations with a same desire if the time interval is short.

In summary, each feature template shown in Table 5.4 has its own meaning in describing certain property of the domain of CoRE. During the course of experiment, it took us several iterations to fine-tune these feature templates so that the domain of CoRE could be properly characterized and depicted.

Based on the prepared training file and template file, a CRF model can be built by CRF++ and stored in a file. However, the feature functions and their associated weights are encoded internally and unfortunately cannot be viewed.

5.5 Desire Inference using Hidden Markov Model

To validate our hypothesis on the advantage of using CRF for desire inference over HMM, I conducted a HMM-based inference experiment as a comparison. The set of training data and test data used to build HMM is same to that mentioned in the previous section. However, the column “Time Interval” (in Table 5.2) is not used because only one class of visible states can be used to build the HMM, so only data of observation have been used. More specifically, the emission probability Matrix O (introduced below) in HMM specify the relations between invisible states (desires) and one class of visible states (observations), and the two types of probability matrixes, Π and T , can only describe desires.

Jahmm [56], a Java implementation of HMM related algorithms, is adopted as the computation tool. Because the Baum-Welch algorithm applied in Jahmm for parameter estimation can find a local minimum of its optimum function only, a critical step in building a HMM model is providing an initial guess of values in the following three basic probability matrixes:

- (1) Π : initial probability distributions of desires. E.g., $P_{\Pi}(d_1)$ represents the

probability of user's desire is d_l at time 0 (the beginning of an observation sequence);

(2) T : matrix of desire transition probability distributions. $P_T(d_i, d_j)$ represents the conditional probability of user's desire is d_j at time $t+1$ given it is d_i at time t , for any $t \geq 0$;

(3) O : matrix of desire-observation emission probability distributions. $P_O(d_i, o_j)$ represents the conditional probability of the observation is o_j given the desire is d_i , at any time point.

Based on the domain of CoRE and the training data, the best way that I found through multiple trials to compute the estimation of the three kinds of probabilities is as the following:

- $P_{\pi}(desire_1) = num(desire_1_seq)/num(all_seq)$, in which $num(desire_1_seq)$ is the number of sequences in which the desire in the first data record is $desire_1$ and $num(all_seq)$ is the number of all sequences;

- $P_T(desire_1, desire_2) = trans_num(desire_1, desire_2) / trans_num(desire_1, anydesire)$, in which $trans_num(desire_1, desire_2)$ is the number of transitions from $desire_1$ to $desire_2$ for all consecutive data records and $trans_num(desire_1, anydesire)$ is the number of transitions from $desire_1$ to any desire;

- $P_O(desire_1, observation_1) = emis_num(desire_1, observation_1) / emis_num(desire_1, anyObservation)$, in which $emis_num(desire_1, observation_1)$ is the number of emissions from $desire_1$ to $observation_1$ in all data records and $emis_num(desire_1, anyObservation)$ is the number of emissions from $desire_1$ to any observation.

Then we can train our HMM based on the initial estimation of the model in Jahmm iteratively. The model is more accurate when the number of iterations is higher. The outcome trained HMM will be used to perform desire inference on the test data.

5.6 Inference Result Analysis in the First-Round Experiment

This step corresponds to Step 3 in our methodology (in section 4.3). Separated from training data, the rest data records are used as test data in the first-round experiment, containing 3,950 data records and 211 experiment sessions. The format of the test data is the same as that of the training data. Based on the built CRF model, desire inference can be performed on the test data using CRF++. Table 5.9 shows a sample of the results of desire inference.

Table 5.9 Format of Desire Inference Result Using CRF++

Test Data			Inferred Desire/ Inference Probability
Observation	Time Interval	Desire	
..... # 0.075707
clickMenuAllPapers	30s	ViewAllPapers	ViewAllPapers/0.925086
clickLogin&LoginGood	10s	ViewAllPapers	ViewAllPapers/0.957425
clickMenuAllPapers	10s	ViewAllPapers	ViewAllPapers/0.982439
.....

Each inference result contains the inferred desire and the inference probability. Meanwhile, the overall inference probability of the output desire sequence for each experiment session is also given, e.g., in Table 5.9, # 0.075707. By examining the inference result, I found that the overall inference probability of the whole desire sequence has a large correlation to the length of the sequence, while the probability of each single output desire can better reflect the inference accuracy. I also found that using the same training data and test data but different feature templates, the inference accuracy will be different. Table 5.10 lists the inference results of using HMM and CRF models with different types of feature templates.

Table 5.10 Inference Accuracy of Different Templates

	Template 1 U01, B01~B04	Template 2 U01~U06	Template 3 U01~U06, B01~B04	Jahmm
%Mislabeling ^a	11.2405% (444/3950)	14.0759% (556/3950)	8.9367% (353/3950)	26.6329% (1052/3950)
Avg-P(Des-Infer) ^b	0.808979	0.749702	0.847904	NA
Avg-P(Seq-Infer) ^c	0.234812	0.0971787	0.286588	NA

^aRatio of mislabeled observations to all observations

^bAverage inference probability of single observations

^cAverage inference probability of experiment sessions

The results in Table 5.10 show that overall CRF has far high inference accuracy than HMM, and it gives better inference results (more accurate) when more meaningful feature templates are used and the domain is described more thoroughly.

To identify users' divergent behaviors that often reflect their new intentions (introduced in section 4.3), it is necessary to choose data records which have high probability to indicate such behaviors in an effective and efficient way. In our expectation, observation records with users' divergent behaviors will probably be labeled with desires which are not consistent with their real desires because the CRF model cannot explain these behaviors very well. However, in practice, since users do not report their desires, it is not possible to detect users' divergent behaviors through comparing the inferred desires with the self-reported ones. One alternative way is to look into the output desires with low inference probability, since there may be some relations between inference probability and inference accuracy. Based on our analysis on data records, I found that basically there was an inverse relationship between inference probability and mislabeling probability, i.e., an observation is more likely to be mislabeled if its inference probability is lower. As shown in Table 5.11, the mislabeling rate (%Mislabeling) is higher for

those observations with lower inference probability. In practice, since we cannot study all data records, we should focus on analyzing observations with low inference probability first.

Another two relations that are useful for further improving the efficiency to filter data records for analyzing divergent behaviors are introduced as following:

1) Consecutively low inference probabilities indicate that mislabeling is more likely occurring (shown in Table 5.12). For example, if the inference probabilities of three consecutive observations are all 0.45 , according to Table 5.12, the probability of all of them are mislabeled is about 44.8% , which is higher than the mislabeling rate for a single observation in the range $[0.4, 0.5)$ shown in Table 5.11. In practice, if two or more consecutive observations are all inferred with low inference probabilities, they should be more likely mislabeled and should be chosen for analyzing.

2) The sharp changes of the inference probability between that of an observation and its neighboring (previous and next) observations indicate it is probably mislabeled (shown in Table 5.13). For example, if the inference probabilities of three consecutive observations are $0.85, 0.4, 0.85$, respectively, the probability of the middle one is mislabeled is 59.64% , which is higher than the mislabeling rate for a single observation in range $[0.4, 0.5)$ in Table 5.11.

The above two relations are used as the principle for filtering data in our study, i.e., we try to find sequences of observations with low inference probability or single observations which has large probability change with its neighbors. Based on the data records characters in our experiment, our filtering mechanism tries to find the following data records for study:

- 1) Two or more consecutive observations with inference probabilities less than 0.6 ;
- 2) One observation whose inference probability is at least 0.2 less than its previous and next observations.

Table 5.11 Mislabeling Rate of Observations in Different Inference Probability Ranges

Inference Probability Range ^a	< 0.1	0.1 ~ 0.2	0.2 ~ 0.3	0.3 ~ 0.4	0.4 ~ 0.5	0.5 ~ 0.6	0.6 ~ 0.7	0.7 ~ 0.8	0.8 ~ 0.9	0.9 ~ 1
%Occurrence ^b	0.10%	1.29%	1.82%	3.22%	3.92%	4.15%	5.19%	6.43%	10.15%	63.72%
%Mislabeling ^c	75%	60.78%	45.83%	44.09%	41.29%	27.44%	19.02%	12.60%	7.48%	0.79%

^aThe range of the inference probability for an observation

^bThe ratio of the number of observations with inference probability in the range to the total number of observations

^cThe ratio of the number of mislabeled observations to the total number of observations with inference probability in this range.

Table 5.12 Mislabeling Rate of Consecutive Observations in Different Probability Ranges

Common Range of Inference Probabilities of Consecutive Observations ^a	Two Consecutive Observations					Three Consecutive Observations				
	< 0.2	< 0.3	< 0.4	< 0.5	< 0.6	< 0.2	< 0.3	< 0.4	< 0.5	< 0.6
%Occurrence ^b	0.46%	1.19%	2.68%	4.92%	7.94%	0.25%	0.66%	1.44%	2.86%	4.83%
%All-Mislabeling ^c	55%	48.08%	47.01%	42.79%	36.02%	54.55%	48.28%	50.79%	44.8%	35.55%

^aThe common range of the inference probabilities of two or three consecutive observations

^bThe ratio of the number of consecutive observation pairs with inference probabilities in the range to the total number of consecutive observation pairs

^cThe ratio of the number of consecutive observation pairs which are all mislabeled to the total number of consecutive observation pairs with inference probabilities in the range

Table 5.13 Mislabeling Rate of Observations in Different Probability Change Ranges

Range of Probability Change ^a	> 0	> 0.1	> 0.2	> 0.3	> 0.4	> 0.5	> 0.6	> 0.7
%Occurrence ^b	18.42%	4.99%	2.45%	1.31%	0.62%	0.16%	0.068%	0
%Mislabeling ^c	17.52%	30.73%	37.38%	42.11%	59.26%	57.14%	66.67%	NA

^aThe common range of inference probability change between an observation and its previous and next observations

^bThe ratio of the number of observations with neighboring inference probability changes in the range to the total number of observation

^cThe ratio of the number of observations which are mislabeled to the total number of observations with inference probability change in the range

The threshold of the values in the above two methods can be determined by the number of total records in practice. For example, people can start with 0.2 in the first method and 0.6 in the second method, and increase the first value or decrease the second value until enough records have been found.

5.7 New-Intention-Detection Case Study

This step corresponds to Step 4 in our methodology (in section 4.3) and system evolution process (in section 4.4). Three new intention detection methods have been applied and demonstrated with illustrative examples based on the first-round experiment results which are shown in below:

1) New Intention Detection Method I

As introduced in the previous section, I focus on studying consecutive observations with low inference probability (< 0.6), or single observations with large inference probability change (> 0.2). These observations are more likely mislabeled, i.e., the CRF model cannot accurately explain these behaviors so they are probably divergent behaviors. Here I give some examples to demonstrate how to detect users' divergent behaviors and use them for system improvement.

a. New_Intention 1: some users often click button *hide the above selection form* to hide the selection form (see Figure 5.3) immediately after entering the page *All Papers*.

As shown in Table 5.12, the observation *clickHideSelection* is labelled with *ViewAllPapers* as the inferred desire with a low probability 0.401886, and its previous observation is also labelled with a low probability desire, while its following observation is labelled with a very high probability desire. In this case, this particular observation (hiding the selection form) can be easily singled out for analysis. To understand such phenomena, I made a

wild guess that the user might think the selection form is cumbersome when he tries to view the information of papers because it is too big and takes up too much screen space.

Two options have been considered to solve this problem: (1) make the selection form initially hidden on the page *All Papers* (users can make it visible by clicking the link show the selection form); (2) show a simplified selection form initially (users can make it complete by clicking the link expand the selection form). To assess which option is better, I did a statistical analysis of users' behaviors after entering the page *All Papers*. There are four kinds of behaviors: a) hide the selection form to view the information of a paper; b) directly view the information of a paper; c) filter papers; d) others (exclude users' aimless behaviors). The number of occurrences of each behavior is: 49, 52, 67, 19, respectively. Because the frequency of behavior c) is highest, it is better to keep the selection form while shrink it to a small size. Therefore, I proposed a modification to the system, that is to simplify the selection form with most frequently search items (key words in the title and publication type) and add a new link expand selection form while initially hide the selection form (see Figure 5.3).

Table 5.14 Example Segment in the Desire Inference Results

Observation	Time Interval	Inference Results
clickMenuAllPapers	10s	ViewAllPapers/0.510326
clickHideSelection	60s	ViewAllPapers/0.401886
clickPaperInfos&PaperID	60s	ViewAPaperInfo/0.986832
clickPaperInfos&PaperID	30s	ViewAPaperInfo/0.995394

b. System_Drawback 1: As shown in Table 5.15, there are two consecutive error messages corresponding to records in row No. 2 and 3. The first error is *ShortLimitation* and the second one is *NoCategory*. Such case is unexpected because the user should have selected the

category when the first error occurs, otherwise the first error message will also contain *NoCategory*. After looking into the original system design, I learnt that the second error occurred because the system cleared users' previous category selection when the first error occurred, however, the user didn't notice such change. The corresponding modification on the system is to let it always keep users' category selection when they submit a comment.

Table 5.15 Example Segment in the Desire Inference Results

Observation	Time Interval	Inference Results
clickSubmitComment&PaperID	60s	SubmitComment/0.883791
clickSubmitComment&ShortLimitation	m	SubmitComment/0.844754
clickSubmitComment&NoCategory	30s	SubmitComment/0.628784
clickSubmitComment&CommentGood	10s	SubmitComment/0.894764

Other examples of users' divergent behaviors detected in our experiment are:

c. System_Drawback 2: Click the button Submit twice or more on the page submit/edit a comment: The reason why users did this might be because the message for successful submission is not clear. Our proposed modification is to use bright color and bold font to make the message more visible (see Figure 5.4);

d. System_Drawback 3: Users consecutively input wrong passwords on the login page and the wrong passwords contain a space. One possible reason is that the user may copy the password with an extra space from somewhere. A possible modification is to give a hint that the password shall contain no space when such case happens.

2) New Intention Detection Method II

Another way to find users' potential new intentions is to study the desire transitions. The following Table 5.16 shows some examples of desire transitions in the first-round experiment:

Table 5.16 Desire Transitions in the First-Round Experiment

Desire Transition	Occurrences	Ratio
UploadPaper → ViewMyPaperInfo ^a / UploadPaper→Any ^b	23/50	46% ^c
SubmitComment → ViewMyCommentInfo / SubmitComment→Any	35/52	67.31%
ViewAPaperInfo → DownloadPaper / Any→DownloadPaper	98/160	61.25%

^aThe desire transition from “UploadPaper” to “ViewMyPaperInfo”

^bThe desire transition from “UploadPaper” to any other desire

^cThe ratio of desire transition a to desire transition b

New intentions can be defined for those frequently occurred desire transitions to make them smoother and more efficient, and on the system can be accordingly modified to simplify users’ operations. For example, for desire transitions *UploadPaper* → *ViewMyPaperInfo* (New_Intention 2) and *SubmitComment* → *ViewMyCommentInfo* (New_Intention 3), new functions can be added to the system to display a link of the submitted paper/comment on the submission page right after the paper/comment is successfully submitted (see Figure 5.4); and for the desire transition *ViewAPaperInfo* → *DownloadPaper* (New_Intention 4), a link can be added on the page of paper information to allow users directly download the paper (see Figure 5.5).

3) New Intention Detection Method III

If an erroneous behavior appears in the observation set for many times (e.g., ≥ 5 times), it can be viewed as a common error. A special case is users’ aimless behaviors that may trigger errors that are not useful for analyzing users’ desires. In this case, CRF is useful to exclude the noisy “error” data, and the aimless behaviors are usually labelled with desire “test” by the CRF model which encodes many “test” behaviors that will be ignored for new intention detection. After removing the noisy data, the occurrences of users’ erroneous behaviors can be easily counted using simple mathematical methods.

Some examples of common users' erroneous behaviors detected in the first-round examples are:

a. New_Intention 5: No file was uploaded when editing the information of a paper. Such erroneous behavior occurs probably because users do not want to upload a file when editing the paper information, which can be considered as a new intention. The corresponding system modification is to allow users to edit the paper information without uploading a file;

b. System_Drawback 4: Number of words in comment details is fewer than the minimum limit when submitting a comment. Users often consecutively encountered this error because they probably have no idea how many words they have typed in. One possible improvement can be that the system shall always display the number of words allowed left;

c. New_Intention 6: Chinese/Japanese comments are not accepted even though they contain enough words, which dues to the fact that the word count function in the system can only count spaces in a comment, which is not suitable for comments written in other languages containing no spaces. One corresponding improvement could be that the system shall be enhanced to support non-English comments;

d. System_Drawback 5: Missing key information (normally DOI, keywords) when uploading a paper. One corresponding improvement could be that the system should give clear tips on the uploading page to tell users what information is mandatory before uploading a paper;

e. New_Intention 7: Non-PDF files are uploaded but not accepted by the system. One corresponding improvement could be that the system is enhanced to support non-PDF files.

Based on the new intentions and system drawbacks I discovered, I made corresponding modification on the system, and the current CoRE system now has evolved to its next version – CoRE Verision II.

Selection Form

Title contains: Authors contains: Publication type: Any

Publisher: Paper type: Any Uploader: Any

Keywords contains: Filter: Any 0.00 Reviewer: Any

Publish date: Between: Jan. 1950 And: March 2015 Upload date: Between: Feb. 5 2014 And: March 3 2015

With review?: Any Last reviewed date: Between: Feb. 5 2014 and March 3 2015

Categories:

- Internet Architecture and Applications
- Software Life Cycle, Evolution, and Maintenance
- Software Testing
- Real-time and Embedded Systems
- Education, Learning and Discourse
- Digital Cities and Public Places
- Network Security
- Requirements Engineering
- Reliability, Metrics, and Fault Tolerance
- Mobile and Pervasive Computing
- Social Networks and Crowd Sourcing
- Network Middlewares, Cloud Architecture and Computing
- Formal Methods
- Trust and Privacy
- Semantic Web
- Big Data Analytics and Knowledge Management
- M2M Networking and Internet of Things
- Software Architecture and Design
- HCI and Usability
- Web Services and Business Process Management
- eHealth and Well-being

Filter

You can choose to [hide the above selection form](#). You will be able to display it again at any moment.

↓

Title contains: Publication type: Any **Filter**

You can choose to [expand the above selection form](#). It will allow you to select more features of papers.

Figure 5.3 Selection forms (upper: selection form in CoRE Version I, lower: selection form in CoRE Version II).

The following information have been stored for the paper Test.
Click [here](#) to view your comment for this paper.

The following information on your paper:
1. Title: Test
2. Authors: Test Test
have been successfully stored in the system. Please click [here](#) to check the paper information.

Figure 5.4 The new link for viewing the submitted comment and for viewing the uploaded paper.

Information on paper 1420804189	
Title:	Layered Approach Using Conditional Random Fields for Intrusion Detection (Download)
Authors:	Kapil Kumar Gupta, Ramamohanarao Kotagiri
Keywords:	Intrusion Detection, Layer Approach, Conditional Random Fields, Network Security, Decision Tree, Naive Bayes

Figure 5.5 The new link for downloading a paper.

5.8 Validation of System Improvement in the Second-Round Experiment

To demonstrate and validate the potential improvement of the system, a second round experiment was done on the evolved system – Core Version II, by following the exactly the same process as what I have done in the first round, which means that I strictly followed Step 1 to Step

4 in section 4.3 for one more iteration. The only difference was that I recruited new participants in the second round to make the comparison between two versions fair and even. In this section, I will not focus on the technical execution of our methodology steps due to the paragraph limit. Instead, I will mainly focus on evaluating the improvement of CoRE version II over its predecessor.

Similar to the data analysis in the first round, a CRF model is built using 2,984 training data records, and it is used for desire inference on a test data set with 3,947 records. The results of desire inference in two rounds are shown in Table 5.17:

Table 5.17 Overall Desire Inference Accuracy

	%Mislabeling	Avg-P(Des-Infer)	Avg-P(Seq-Infer)
Second Round	3.9524% (156/3947)	0.905067	0.314128
First Round	8.9114% (352/3950)	0.848429	0.28694

As we can see, the overall inference probability and inference accuracy in the second-round experiment are significantly improved compared with the inference results in the first round. Some example system improvements and the evaluation of these improvements are described as below:

1) Simplified selection form (New_Intention 1, Figure 5.3)

To evaluate the benefit of the simplified selection form in the new system, I did a statistical analysis of users' behaviors after entering the page *All Papers* on which the selection form is located. There are four kinds of behaviors: a) use the simplified selection form to filter papers; b) expand the selection form to filter papers; c) view the information of a paper; d) others (exclude users' aimless behaviors). The number of occurrences of each behavior is: 54, 41, 63 and 30 respectively. Since users used the simplified selection form more often than they

expanded the selection form, we can draw the conclusion that it is more suitable to display the simplified selection form on the page *All Papers* instead of the expanded version.

2) The link of the newly submitted comment in the message of successful submission/editing, and a link of the newly uploaded paper in the message of successful uploading/editing (New_Intention 2&3, Figure 5.4).

Table 5.18 shows the use of the new links when users upload/edit a paper or submit/edit a comment, and it can be seen that the new link has been frequently used. Meanwhile, consecutive Submit operations didn't occur at all in the new system, which also demonstrate the effectiveness of the modification.

Table 5.18 The Use of New Links for Viewing Paper Information

Observation	Occurrences	Ratio
Upload Paper → View Paper Info ^a / Upload Paper → Any ^b	28/32	87.5%
Submit Comment → View Paper Info / Submit Comment → Any	82/92	89.13%
Edit Paper → View Paper Info / Edit Paper → Any	6/9	66.67%
Edit Comment → View Paper Info / Edit Comment → Any	16/22	72.73%

^aUse the new link to view paper info after uploading a paper

^bAny behavior after uploading a paper

3) Download link on paperinfo page (New_Intention 4, Figure 5.5)

The observed users' behaviors in the second-round experiment show that users prefer the new link to the old one after viewing the paper information, considering that the total use count of the new link is 146, while that of the old link is 60.

To demonstrate the overall system improvement, I focused on four major user tasks/operations, which are uploading/editing paper, submitting/editing comment, and evaluated users' performance on those tasks in both rounds of experiments. The comparison result is shown

in the Table 5.19. As we can see, the success rate of four tasks all increased in the second-round experiment, while the time cost and the error occurrence rate, especially the consecutive error occurrence rate, significantly decreased.

Table 5.19 Comparison Between Two Rounds Experiments

Task	Round	# Of Occur.	Avg. Time Spent (s)	Successful	Errors^a	Consecutive Errors^b
Upload Paper	1 st	55	387.87	38 (69.09%)	28 (50.91%)	14 (25.45%)
	2 nd	32	173.57	31 (96.88%)	1 (3.125%)	0
Submit comment	1 st	42	287.26	38 (90.48%)	23 (54.76%)	15 (35.71%)
	2 nd	100	207.79	92 (92%)	49 (49%)	6 (6%)
Edit Paper	1 st	16	47.81	8 (50%)	5 (31.25%)	0
	2 nd	9	47.33	9 (100%)	0	0
Edit comment	1 st	29	105.38	20 (68.97%)	0	0
	2 nd	12	64.25	10 (83.33%)	0	0

^aThe times of at least one error occurs when submitting/editing a paper/comment

^bThe number of occurrences of consecutive errors

Overall speaking, through the evolution from CoRE version I to II, the system usability has been improved, which means the evolution based on our methodology was successful.

5.9 Summary of the Experiment

Based on the above demonstration of the execution of the experiment and the analysis of the results, we now can revisit those three hypotheses proposed at the beginning of this chapter, and see if each of them is valid or not.

- 1) Desire inference with CRF model:

According to inference accuracy results of the first round experiment (Table 5.10, section 5.6), CRF model is able to deliver highly accurate desire inference result when appropriate feature templates have been designed and applied. The actual inference accuracy rate is 91%, which easily meets our expectation ($> 90\%$). Additionally, the average inference probability (confidence) is about 0.85. The combination of high inference accuracy and confidence prove the fact again that CRF model is good at sequential labelling. On the contrary, using the same training and test data sets, HMM fell short on the accuracy aspect as theoretically expected, with less than 74% accuracy rate.

2) New intention & system drawback detection:

According to our case study in section 5.7, 7 new intentions and 5 system drawbacks have been identified for CoRE Version I using the three newly proposed methods (in section 4.3). Same for CoRE Version II, our methods are effective to detect new intentions in the domain of CoRE. For example, a phenomenon I discovered through desire transition analysis (Method II) is that the user often reviews or downloads paper one by one after filtering papers using the selection form. Since these consecutive Download Paper desires occur frequently, I proposed to introduce a compound desire called Download All Papers by adding a link for downloading all selected papers at once.

The above results and example also demonstrate that our methodology is able to continuously explore users' new intentions and enhance the system to adapt to the ever-changing users' requirements.

3) Successful and rapid system evolution:

Based on our analysis results in section 5.8, the CoRE system successfully evolved from version I to version II, and the evolution process was efficient. In each round of our experiment, I

collected 60 users' behavioral and system contextual information for one month, and got around 10,000 raw data records. And then it took about one month for just one domain expert to process & analyze data, infer desires & detect new intentions, and eventually evolved our experiment system. However, in practice, I believe that our methodology can be even more efficient, especially for those systems or applications with a large user base. For example, it might take Facebook only a few seconds to capture the same amount of raw data. On the other hand, there might be new problems/issues emerged, for example, Big Data issue, which is beyond the discussion of this thesis. However, as one of the technical merits of our methodology, I use real system usage data to drive and enable system evolution, which makes the whole process pretty straightforward. As what have been shown in section 5.7, those newly detected intentions and system drawbacks are quite intuitive and self-explained. Thus, it won't take much intellectual effort to figure out the corresponding new requirements and system remedies.

In summary, our two-round, exploratory experiment was successful, in the sense that all hypotheses have been proven valid, and our proposed methodology has been demonstrated to be effective and efficient.

5.10 Threats to Validity

In this section, I will discuss some potential threats to the validity of our proposed methodology and experiment from the following perspectives:

- 1) Threats to construct validity: concerns regarding the design of my methodology and the measurement of my metrics;
- 2) Threats to internal validity: concerns regarding alternate explanations for the experimental results;
- 3) Threats to external validity: concerns regarding the generalizability of my results.

5.10.1 Threats to Construct Validity

One potential concern about the validity of my methodology is the theoretical foundation (section 3.1 & 3.6), in which the computational models for situation and intention have been defined. Although I carefully and thoroughly characterized and described the human-centric context-aware domains with certain causal relationships among users' desire, actions and relevant context values by making a number of assumptions and axioms, some exceptional cases might not be covered or might newly emerge along with the evolution of these domains. Simply said, any threat to the validity of theoretical foundation can jeopardize our experiment.

Besides, there are also a couple of concerns regarding the measurement of our metrics:

1) Interaction of normal operations and reporting desires

To directly validate our desire inference results, I asked participants to report their real-time desires while operating on the system, which is considered as a necessary part of our experiment. However, such additional task (reporting desires) might interfere with participants' normal thinking process and might further influence their understanding of the system. An obvious example would be that the dropdown desire list on each webpage of CoRE contains all the predefined desires related to the system, and participants may be able to learn from it, and consequently adjust their behaviors. Although such phenomenon is undesired, the value of acquiring participants' self-reported desires significantly outweighs its side effect.

2) Hypothesis guessing

Due to our local IRB regulations, I need to fully introduce and explain our experiment to all participants, including our basic experiment motivation and detailed procedure. Although I tried to be very brief on our experiment goals and technical methods, participants might be able to guess what hypothesis I want to validate, and even what data can help prove those hypothesis

valid. Such possibility cannot be eliminated, since the majority of the participants are majored in computer science, computer engineering, or related fields. However, I doubt the possible impact of such phenomenon on the quality of our data, because the CoRE system appears to be a regular online library, and looks quite intuitive and familiar to most of the participants, and they could finish most of the tasks without thinking too much about the system and the experiment itself. Therefore, I expect the impact of hypothesis guessing to be minimal in our experiment.

5.10.2 Threats to Internal Validity

For internal validity, I will look into whether there is sufficient evidence to support the conclusions of our experiment, specifically in the following aspects:

- 1) Design of comparison model (HMM)

To demonstrate that CRF has good performance in desire inference, and I designed a comparison model, HMM, and evaluated both methods' desire inference results side by side in Table 5.10 (section 5.6). However, since HMM is designed differently from CRF, it is hard to construct the comparison absolutely fair and even. To be specific, as introduced in section 5.5, one critical step for building a HMM is to design the initial estimation of values in the three matrixes, while the critical step for building a CRF model is to design the feature templates. Although the comparison result shows CRF outperformed HMM, the possibility of not having the best HMM still cannot be eliminated. What I could do was to try with different HMMs, and pick the best one to compare with CRF. But I didn't know if the one I chose was the real best one or not. Such issue commonly exists in many related statistical studies, and our way to handle it is considered as acceptable.

- 2) Prediction of new requirements

The new requirements revealed from the analysis of divergent behaviors were mainly based on our subjective judgement, and were validated through usability analysis between two versions of systems indirectly, instead of directly inquiring the users. Therefore, it is still debatable whether our predicted new requirements were in fact the users' new requirements, or just our own preferred system evolution path. However, when evolving a real-world system, engineers may face the same problem, and one way to truly validate newly found requirements is to evaluate through usability analysis of the new system. But it must be admitted that improvement of system usability may due to different reasons, which might not be the implementation of the specific new requirements. For example, the time spent for uploading a paper (Table 5.19) decreases significantly may because some previously required information has been removed, such as DOI. So a reasonable conclusion that I can draw for the evolution of CoRE is that its overall system usability has been improved by implementing new requirements and necessary remedies.

3) Arbitrary design of desire granularity

To build the domain knowledge, a prerequisite is to define and enumerate all the anticipated desires. The quality of the predefined desire set is crucial for the accuracy of desire inference. Especially the granularity of desire must be properly designed. If the desire is too fine, it will create too many labels for CRF to choose from, and will probably confuse the CRF model. Or if the desire is too coarse, it may not be able to generalize the causal relations among desire, actions, and context values. In our experiment, the desires in the domain of CoRE were relatively easy to define. However, I haven't tested other domains for the feasibility and the difficulty of designing a set of appropriate known desires.

4) Statistical conclusion validity

In essence, I use statistical methods (Table 5.11, 5.12, 5.13) to identify candidate observations for new intention detection, and we basically depends on the quality and characteristics of sample data (mislabelled records). When the sample space is bigger, our methodology is more likely to work well. However, when the sample space is too small or has too much noise (false positive), our methodology, or any method based on statistical analysis, may not give an ideal result. One example is that for a matured system (domain), there might be only few new intentions, which correspond to only few observations. While having such a small target mixed with a few noisy data, it is hard to elicit the real observations of our interests.

5.10.3 Threats to External Validity

In our methodology, I described its application domain as a human-centric context-aware domain. And our experiment was conducted on a system which is supposed to be a representative application. However, as I stated in Assumptions 1, 2 and 3, the domain should be context aware, and belong to a single agent who has a single desire at any instant. If an application domain doesn't have these attributes, it is not applicable for our methodology. For example, if users interact frequently, and their actions directly influence each other's desire, their behaviors will be very hard to describe and desire inference is not able to perform (using our methodology). However, I do believe that our methodology has broad application prospects because many applications can be characterized as human-centric context-aware domains as I described in chapter 3. Some example systems to apply our methodology are listed as the followings.

1) Dynamic web-based systems' server side is responsible for processing and responding to users' requests, where monitoring programs can be deployed to capture users' operations, inputs, submissions and contents on the pages.

2) Mobile applications which can sense users' physical status and environment conditions, and adapt their behavior accordingly. In fact, some research work has been conducted to effectively select services for adaptation according to the user's current context [57].

3) Home automation [58] is another promising application area, especially for smart home for elderly and disabled [59], in which most of users' behaviors can be monitored and users have eager demand to increase the quality of life without caregivers or institutional care.

CHAPTER 6. DISCUSSIONS & CONCLUSIONS

This thesis presents an effective method of applying Conditional Random Fields as the mathematical foundation to infer human desires based on observations of their actions and relevant environmental context values, and further explore their new intentions for driving system evolution. With an explorative experiment, I show that the accuracy of desire inference is pretty high ($> 90\%$) by using CRF compared with the result of using HMM (75%). Furthermore, the three methods for detecting new intentions (section 4.3) have been validated to be effective to obtain users' new potential intentions and further reveal their new requirements on the system or system drawbacks that are useful for system evolution. The experiment results verified our hypothesis about fast discovery of new requirements and system evolution based on our methodology.

However, there are some limitations of our work. Besides the application scope delimited by Assumption 1, 2 and 3 (section 3.1), our methodology is only able to capture users' new functional requirements not non-functional requirements such as high usability, fast response time, etc. In addition, the analysis of potential new intentions and users' new requirements still needs a lot of human efforts and may not consistently give the best result. Therefore, the gap between desire inference and new intention detection presents steep research challenges for us to tackle in the future:

- 1) The methodology proposed in this thesis can only automatically detect users' potential desires related to new intentions and system drawbacks. The work of eliciting and verifying new intentions and system drawbacks, as well as designing and implementing new system functionalities would require human efforts. From a knowledge engineering perspective,

it is worthy being further investigated as to how to acquire design wisdom from raw data analytics.

2) In my methodology, I restrict the domain as a single agent domain, which is unable to characterize multiple agent domains, especially in which agents frequently interact with each other. To extend the applicability of our methodology, there are some interesting research questions to be answered. For example, how to characterize the causal relationships among agent's desires, actions, and context values or can CRF encode such causal relationships among other research questions.

3) In my experiment, I demonstrated how to apply our methodology onto a web-based system. In later sections, I also discussed the feasibility of applying our methodology on different services. To verify that the method is really applicable for other systems, experimental validation is still much needed. Some valuable candidates are mobile applications, smart home systems, etc., which are becoming prevalent nowadays with rapidly changing user requirements.

REFERENCES

- [1] K.H. Bennett and V.T. Rajlich, "Software Maintenance and Evolution: a Roadmap," *Proc. Conf. The Future of Software Engineering*, ACM, pp. 73-90, 2000.
- [2] I. Borne, S. Demeyer, and G.H. Galal, "Object-Oriented Architectural Evolution," *Object-Oriented Technology ECOOP'99 Workshop Reader Lecture Notes in Computer Science*, vol. 1743, pp. 57-79, 1999.
- [3] D. Rowe and J. Leaney, "Evaluating evolvability of computer based systems architectures - an ontological approach," *Proc. Int'l Conf. and Workshop on Engineering of Computer-Based Systems*, pp. 360- 367, 1997.
- [4] H. Ming, K. Oyama, and C. Chang, "Human-Intention Driven Self Adaptive Software Evolvability in Distributed Service Environments," *Proc. 12th IEEE Int'l Workshop Future Trends of Distributed Computing Systems (FTDCS '08)*, pp. 51-57, 2008.
- [5] A. Iliasov and A. Romanovsky, "CAMA: Structured Communication Space and Exception Propagation Mechanism for Mobile Agents," *Proc. ECOOP Workshop Exception Handling in Object Oriented Systems: Developing Systems That Handle Exceptions*, pp. 75-87, 2005.
- [6] H.P. Breivold, I. Crnkovic, R. Land, and S. Larsson, "Using dependency model to support architecture evolution," *Proc. 4th International ERCIM Workshop on Software Evolution and Evolvability (Evol'08) at the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE'08)*, pp. 82-91, 2008.
- [7] M.M. Lehman, J.F. Ramil, P. Wernick, D.E. Perry, and W.M. Turski, "Metrics and Laws of Software Evolution—the Nineties View," *Proc. Fourth Int'l Software Metrics Symposium*, pp. 20-32, 1997.
- [8] H.P. Breivold, I. Crnkovic, and M. Larsson, "Software architecture evolution through evolvability analysis," *Journal of Systems and Software*, vol. 85, Issue 11, pp. 2574–2592, November 2012.
- [9] C.L. Nehaniv and P. Wernick, "Introduction to software evolvability," *Proc. Third Int'l IEEE Workshop on Software Evolvability*, pp. 6-7, 2007.
- [10] S. Robertson and J. Robertson, *Mastering the Requirements Process: Getting Requirements Right (3rd Edition)*, Chapter 14: Requirements and Iterative Development, Addison Wesley, 2012.
- [11] M. Marschall, "Transforming a Six Month Release Cycle to Continuous Flow," *Proc. Assoc. Geographic Information Laboratories Europe Conf. (AGILE '07)*, pp. 395-400, 2007.

- [12] J. Gorinsek, S. Van Baelen, Y. Berbers, and K. De Vlamincx, "Managing Quality of Service during Evolution Using Component Contracts," *Proc. ETAPS 2003 Workshop Unanticipated Software Evolution (USE '03)*, pp. 57-62, 2003.
- [13] O. Saliu and G. Ruhe, "Supporting Software Release Planning Decisions for Evolving Systems," *Proc. 29th IEEE/NASA Software Eng. Workshop (SEW-29)*, pp. 14-26, 2005.
- [14] C. Salinesi and A. Etien, "Compliance Gaps: A Requirements Elicitation Approach in the Context of System Evolution," *Proc. 9th International Conference on Object-Oriented Information Systems (OOIS 2003)*, pp. 71-82, 2003.
- [15] W. Jiang, H. Ruan, L. Zhang, P. Lew, and J. Jiang, "For User-Driven Software Evolution: Requirements Elicitation Derived from Mining Online Reviews," *Proc. 18th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining (PAKDD 2014)*, pp. 584-595, 2014.
- [16] C. Chang, K. Oyama, H. Jaygarl, and H. Ming, "On Distributed Run-Time Software Evolution Driven by Stakeholders of Smart Home Development," *Proc. Second Int'l Symp. Universal Comm. (ISUC '08)*, pp. 59-66, 2008.
- [17] R. Laddaga, "Self Adaptive Software—Problems and Projects," *Proc. Int'l Workshop Software Evolvability (SE '06)*, pp. 3-10, 2006.
- [18] J. Xia, C. Chang, T. Kim, H. Yang, R. Bose, and S. Helal, "Fault-Resilient Ubiquitous Service Composition," *Proc. Third IET Int'l Conf. Intelligent Environments (IE '07)*, pp. 108-115, 2007.
- [19] C. Chang, H. Jiang, H. Ming, and K. Oyama, "Situ: A Situation-Theoretic Approach to Context-Aware Service Evolution," *IEEE Trans. on Services Computing*, vol. 2, no. 3, July-September, 2009.
- [20] L. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proc. IEEE*, vol. 77, no. 2, pp. 257-286, 1989.
- [21] C. Sutton and A. McCallum, *An Introduction to Conditional Random Fields for Relational Learning*. MIT Press, 2006.
- [22] J. Lafferty, A. McCallum and F. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," *Proc. of 18th International Conference on Machine Learning*, Morgan Kaufmann, pp. 282-289, 2001.
- [23] T. Mens and S. Demeyer, *Software Evolution*. Springer-Verlag Berlin Heidelberg, 2008.
- [24] IEEE Std. 610.12-1990, *IEEE Standard Glossary of Software Engineering Terminology*, IEEE, New York, 1991.

- [25] B.P. Lientz and E.B. Swanson, *Software Maintenance Management, A Study of the Maintenance of Computer Application Software in Data Processing Organizations*. Addison-Wesley, Reading MA, 1980.
- [26] P.I. Okwu and I.N. Onyeje, “Software Evolution: Past, Present and Future,” *American Journal of Engineering Research (AJER)*, vol. 03, Issue 05, pp. 21-28, 2014.
- [27] Ligu Yu and Alok Mishra, “An Empirical Study of Lehman's Law on Software Quality Evolution,” *International Journal of Software and Informatics*, vol. 7, Issue 3, pp. 469-481. 2013.
- [28] S. Liaskos, S. McIlraith and S. Sohrabi, “Representing and reasoning with preference requirements using goals,” Technical report, Dept. of Computer Science, University of Toronto, 2006.
- [29] Thomas Keller, “Contextual Requirements Elicitation - An Overview,” Seminar in Requirements Engineering, Department of Informatics, University of Zurich, 2011.
- [30] G.E. Kniessel and R.E. Filman, “Unanticipated Software Evolution,” *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 17, Issue 5, pp. 307–377, 2005.
- [31] V.J. Hodge and J. Austin, “A Survey of Outlier Detection Methodologies,” *Artificial Intelligence Review*, vol. 22, Issue 2, pp. 85-126, 2004.
- [32] L. Getoor and B. Taskar, *An Introduction to Conditional Random Fields for Relational Learning (Edition 1)*. MIT Press, pp. 93–127, 2007.
- [33] E. Chen, “Introduction to Conditional Random Fields,” Retrieved from <http://blog.echen.me/2012/01/03/introduction-to-conditional-random-fields/>, Sep. 2014.
- [34] T. Cohn, “Efficient inference in large conditional random fields,” *Proc. the 17th European conference on Machine Learning (ECML'06)*, pp. 606-613, 2006.
- [35] H.M. Wallach, “Efficient training of conditional random fields,” Master’s thesis, University of Edinburgh, 2002.
- [36] A. McCallum and W. Li, “Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons,” *Proc. the seventh conference on Natural language learning at HLT-NAACL (CONLL '03)*, vol. 4, pp 188-191, 2003.
- [37] A. Culotta, R. Bekkerman, and A. McCallum, “Extracting social networks and contact information from email and the web,” *First Conference on Email and Anti-Spam (CEAS)*, Mountain View, CA, 2004.

- [38] F. Peng, F. Feng, and A. McCallum, “Chinese segmentation and new word detection using conditional random fields,” *International Conference on Computational Linguistics (COLING)*, pp. 562–568, 2004.
- [39] F. Peng and A. McCallum, “Accurate information extraction from research papers using conditional random fields,” *Human Language Technology Conference and North American Chapter of the Association for Computational Linguistics (HLT-NAACL)*, pp. 963-979, 2004.
- [40] Y. Liu, J. Carbonell, P. Weigele, and V. Gopalakrishnan, “Protein fold recognition using segmentation conditional random fields (SCRFs),” *Journal of Computational Biology*, vol. 13, no. 2, pp. 394–406, 2006.
- [41] K. Sato and Y. Sakakibara, “RNA secondary structural alignment with conditional random fields,” *Bioinformatics*, vol. 21, pp. 237–242, 2005.
- [42] X. He, R.S. Zemel, and M. A. Carreira-Perpinian, “Multiscale conditional random fields for image labelling,” *Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 695-702, 2004.
- [43] H. Xie, L. Liu, and J. Yang, “i*-Prefer: Optimizing Requirements Elicitation Process Based on Actor Preferences”, *Proc. 24th ACM Symposium on Applied Computing*, pp. 347-354, 2009.
- [44] A. Quattoni, M. Collins, and T. Darrell, “Conditional random fields for object recognition,” *Advances in Neural Information Processing Systems (NIPS)*, pp. 1097–1104, 2005.
- [45] K.K. Gupta, B. Nath, and R. Kotagiri, “Layered Approach Using Conditional Random Fields for Intrusion Detection,” *IEEE Trans. on Dependable and Secure Computing*, vol. 7, Issue 1, pp. 35 – 49, 2010.
- [46] Y. Shen, J. Yan, S. Yan, L. Ji, N. Liu, and Z. Chen, “Sparse hidden-dynamics conditional random fields for user intent understanding,” *Proc. the 20th international conference on World wide web (WWW '11)*, pp. 7-16, 2011.
- [47] O. Brill and E. Knauss, “Structured and Unobtrusive Observation of Anonymous Users and their Context for Requirements Elicitation,” *Proc. IEEE 19th International Requirements Engineering Conference*, pp. 175–184, 2011.
- [48] A. Alkhanifer and S. Ludi, “Towards a Situation Awareness Design to Improve Visually Impaired Orientation in Unfamiliar Buildings: Requirements Elicitation Study,” *Proc. IEEE 22nd International Requirements Engineering Conference*, pp. 23-32, 2014.
- [49] H. Xie, C. Chang, H. Ming, and K. Lu, “The Concepts and Ontology of SiSL: A Situation-Centric Specification Language”, *Proc. Computer Software and Applications Conf. Workshops (COMPSACW '12)*, pp. 301-307, 2012.

- [50] L.T.F. Gamut, “Logic, Language, and Meaning,” *Intensional Logic and Logical Grammar*, vol. 2, Chicago, IL: University of Chicago Press, 1991.
- [51] S.B. Kotsiantis, “Supervised Machine Learning: A Review of Classification Techniques,” *Proc. of the 2007 conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*, pp. 3-24, 2007.
- [52] T. Kudo, “CRF++: Yet Another CRF toolkit,” Retrieved from <http://taku910.github.io/CRFpp>, May 2014.
- [53] X.H. Phan, L.M. Nguyen, and C.T. Nguyen, “FlexCRFs: Flexible Conditional Random Fields,” Retrieved from <http://flexCRFs.sourceforge.net>, May 2014.
- [54] V. Prabhakaran, A.C. Arpaci-Dusseau, and R.H. Arpaci-Dusseau, “Analysis and Evolution of Journaling File Systems,” *Proc. the USENIX Annual Technical Conference (ATEC '05)*, pp. 8-23, 2005.
- [55] P. Rigaux, “The MyReview System,” Retrieved from <http://myreview.sourceforge.net>, Jan. 2014.
- [56] J.M. François, “An implementation of Hidden Markov Models in Java,” Retrieved from <https://code.google.com/p/jahmm>, Dec. 2014.
- [57] G.S. Thyagaraju and U.P. Kulkarni, “Design and Implementation of User Context aware Recommendation Engine for Mobile using Bayesian Network, Fuzzy Logic and Rule Base,” *International Journal of Computer Applications*, vol. 40, No.3, pp. 47–63, 2012.
- [58] Abi Research, “1.5 Million Home Automation Systems Installed in the US This Year,” Retrieved from <https://www.abiresearch.com/press/15-million-home-automation-systems-installed-in-th>, Mar. 2015.
- [59] P. Harmo, T. Taipalus, J. Knuuttila, J. Vallet, and A. Halme, “Needs and solutions - home automation and service robots for the elderly and disabled,” *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2005)*, pp. 3201-3206, 2005.
- [60] App Annie, “Decision-making platform for the entire mobile app economy,” Retrieved from <https://www.appannie.com>, Mar. 2015.
- [61] G. Weiss, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 2000.
- [62] R. Kelley, A. Tavakkoli, C. King, M. Nicolescu, M. Nicolescu, and G. Bebis, “Understanding Human Intentions via Hidden Markov Models in Autonomous Mobile Robots,” *Proc. Third ACM/IEEE Int’l Conf. Human Robot Interaction (HRI '08)*, pp. 67-374, 2008.