2016

Trend Analysis of Belief-State History with Discrete Wavelet Transform for Improved Intention Discovery

VIjaya Krishna Mulpuri
University of Windsor

Trend Analysis of Belief-State History with Discrete Wavelet Transform for Improved Intention Discovery

by

Vijaya Krishna Mulpuri

A Thesis
Submitted to the Faculty of Graduate Studies
Through Computer Science
In Partial Fulfillment of the Requirements for
The Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada

2016

Trend Analysis of Belief-State History with Discrete Wavelet Transform for Improved Intention Discovery

by

Vijaya Krishna Mulpuri

APPROVED BY:

_____
Dr. Christine Thrasher
Faculty of Nursing

_____
Dr. Scott Goodwin
School of Computer Science

_____
Dr. Xiaobu Yuan, Advisor
School of Computer Science

September 21, 2016

DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

Software Product Lines (SPL) have emerged as a new paradigm of software development. By means of mass production of customized software products, SPL has the potential to significantly reduce development time and cost while improving the quality of software systems. Currently, there is still a severe shortage of tools that support the decision-making process for software clients to interactively "order" software products due to the difficulty of software customization, especially via dialogue in natural language. While most of the existing approaches use POMDP-based dialogue management, this thesis research proposes to introduce historical information of belief states into the POMDP model and to analyze its trend with discrete wavelet transformation (DWT). Accordingly, a new algorithm is developed to improve the accuracy of intention discovery with trend analysis, and to reduce the dialog length by switching POMDP policies between contextual control modes according to the anticipated knowledge of different users. The efficiency and accuracy of the proposed method are examined by experiments with simulation.

DEDICATION

To the almighty god, my mom (M. Subhadra Devi), my grandfather (G. Anatharamaiah) and all my friends for their belief and support.

ACKNOWLEDGEMENT

I would like to take this opportunity to thank my supervisor Dr. Xiaobu Yuan for his encouragement and support in presenting this Thesis work. My ultimate gratitude goes to him for contributing his suggestions and ideas during my research. His insightful feedback and instructions made it possible for me to accomplish this work.

I would like to acknowledge my thesis committee members Dr. Christine Thrasher and Dr. Scott Goodwin whose suggestions and recommendations greatly improved the quality of this work. I would like to thank them for spending their valuable time providing feedback about thesis throughout my proposal and defense.

My special thanks goes to my parents and my family for their patience and love they provided to me during all times. I express my deep appreciation to my all friends for their motivation and moral support they provided during all stages of my thesis work.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

LIST OF ABBREVIATIONS/SYMBOLS

COCOM	Contextual Control Model
CWT	Continuous Wavelet Transformation
DWT	Discrete Wavelet Transformation
ECA	Embodied Conversational Agent
FT	Fourier Transformation
HCI	Human Computer Interaction
MDP	Markov Decision Process
POMDP	Partially Observable Markov Decision Process
RE	Requirements Elicitation
SPL	Software Product Line
STFT	Short-Time Fourier Transformation
TFR	Time-Frequency Representation of a Signal
WT	Wavelet Theory

CHAPTER 1

INTRODUCTION

Requirements Engineering aims to define, document and maintain the software requirements and it is the most vital phase in Software Engineering (SE). It's a complex exercise that considers product demands from a vast number of viewpoints, roles, responsibilities, and objectives [20]. Requirements Elicitation (RE) is a part of Requirements Engineering where requirements of a product/system are gathered from users, clients, and stakeholders [12]. A lot of effort [32] [37] has been put in this area of research and many techniques were identified to reduce errors and make the elicitation process work more efficiently. Consider the following example, a software company gets following requirements from Client A and Client B as illustrated in the diagram.

Client A: {Requirement A1, Requirement A2, Requirement A3}

Client B: {Requirement B1, Requirement B2, Requirement B3}

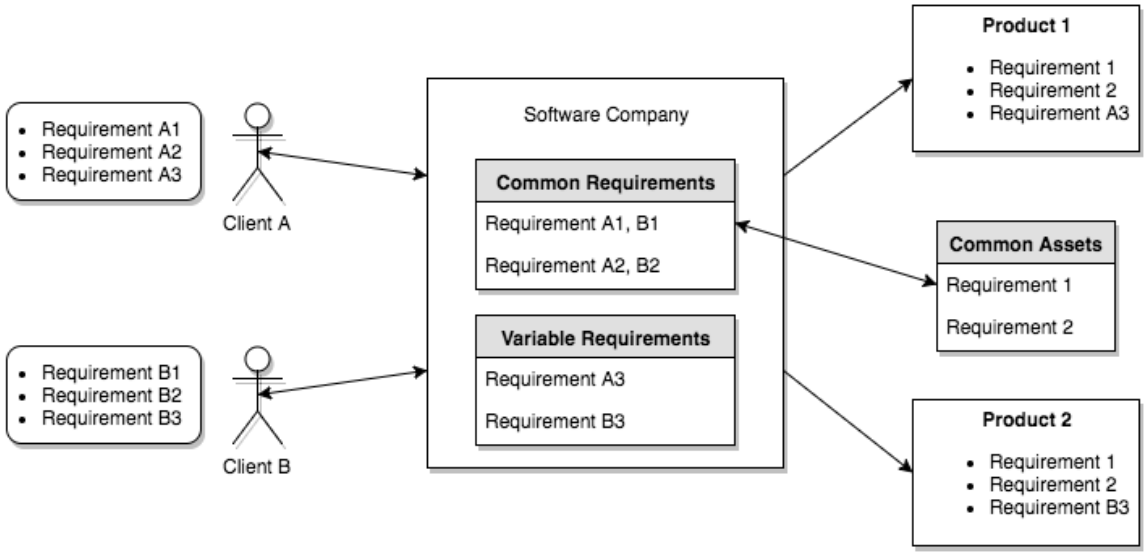


Figure 1.1: Example of Software Product Line

Software Company analyzes the requirements and found that Requirements A1, A2 are same as B1, B2 respectively and A3, B3 were different. By developing common assets library consisting of Requirements A1, B1 as Requirement 1 and Requirements A2, B2 as Requirement 2 and using this common assets library in the development process of Products 1 and 2 will save development time and cost. This technique is known as Software Product Line (SPL) Engineering.

1.1 Software Product Line Engineering

In the course of time, there has been a significant change in the ways of production of software applications due to the increase in demand for mass production of customized software. Software produced are usually denoted either as Individual or Standard Software products. There are many drawbacks of these software produced, like for example Individual software though adaptable to changes are expensive whereas standard software products lack sufficient diversification. Software Product Line (SPL) engineering has emerged as a new paradigm for the development of software applications using reusable software assets, tailored to the individual customer needs [22]. By reusing the common set of core assets in a prescribed way, SPL improves the software customization process by reducing the development cost, time and enhancing the quality. Organizations of all types and sizes can implement SPL strategies and the benefits when implemented skillfully are as follows: [13]

- Productivity is improved by 10x
- Quality is increased by 10x
- Cost is decreased by 60%
- Time to market is decreased by as much as 98%

- Labor needs are decreased by 87%

On the other hand, there were several attempts [11] [17] made to automate requirement elicitation process in SPL engineering. Here automation means gathering requirements and automatically looking for the relevant services/modules available in the common assets library or not to implement the concepts of SPL Engineering. For gathering requirements, automation requires an interactive tool/dialog which can guide humans through the elicitation process by taking necessary inputs or interactions.

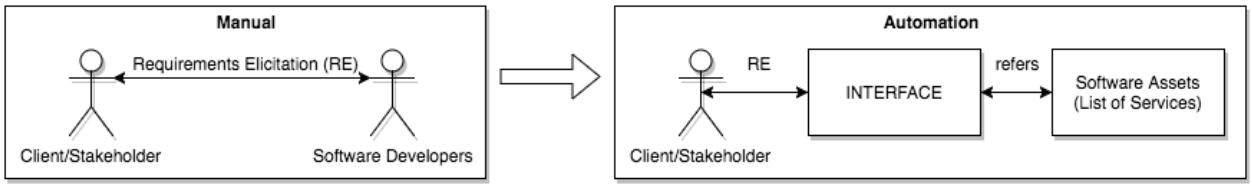


Figure 1.2: An Idea to Automate RE in SPL

1.2 Limitations in Automating RE in SPL using ECA

An Embodied Conversational Agents (ECA) is a computer-generated character with 2D or 3D human object, with human-like appearance and lifelike behavior that answers questions and performs tasks for the user through conversational, natural language-style dialogs [6]. Various approaches [9] [34] have been developed using ECA as an interface to gather requirements in natural language and assist the user accordingly. However, managing the complexity and variability of product features inherent in software product lines is very challenging [11]. In addition, a supporting tool for directing the automatic and interactive product customization is still lacking [25]. According to Mimoun et al. [21] two important aspects which make automation challenging are as follows:

a. *Appearance* – In the process of automation using ECA, we replace human-human interaction in requirement elicitation with human-ECA interaction. So the appearance of ECA should be realistic (human-like). For example, having features like animated speech with lip movement and facial expressions; eye, head and body movements to realize gestures; express emotions; and perform actions or display listening or thinking postures makes them realistic [18]. Also, it is evident that face-to-face interaction allows much richer information exchange than other means of communication [5] [26].

b. *Intelligence* – The automation process involves understanding the requirements in the form of natural language from the user and to make optimized decisions which expedite user goals.

Intelligence aspect is the primary focus of this thesis. Precisely, the thesis focuses on understanding the requirements of the user thereby improving the accuracy of intention discovery and anticipating the user knowledge level to make optimized decisions thereby reducing the dialog length. This demands an advanced algorithm which makes the automation challenging and worth solving.

1.3 The Problem Statement

1.3.1 What is User Intention Discovery? Why improve it?

Generally, intention means the thing intended; aim or plan. An analysis to understand what users aim or plan or actually intended about the requirement is known as User Intention Discovery. For example, when user provides requirement as follows:

User Requirement> *I need search by keyword feature for my bookstore*

System Response> *Would you like search to be a broad match or exact match?*

System (ECA) should understand that the user is looking for the "Search by keyword" feature and checks if the service/module is available in the common assets or not. Based on the service availability, System (ECA) will provide necessary response to understand more about the specified requirement thereby improving the user intention discovery. By improving the accuracy of user intention discovery we are making the automation process better.

1.3.2 What is dialog length? How to optimize it?

Automation of RE in SPL involves communication between user and system (ECA). The number of conversations exchanged or interactions made between user and system (ECA) is known as dialog length. The dialog length can be optimized by anticipating the user knowledge level. For Example, consider following cases:

Case 1: Consider an Expert User interacting with System (ECA)

User> *I need search by keyword feature for my bookstore*

System> *Would you like search to be a broad match or exact match?*

User> *exact match is what I am looking for*

Case 2: Consider a Naive User interacting with System (ECA)

User> *I need search by keyword feature for my bookstore*

System> *Would you like search to be a broad match or exact match?*

User> *What do you mean by broad and exact match?*

System> *Broad match means …. and exact match means …. What would you like?*

In Case 1, the user is an expert and has knowledge about what is a broad match and exact match. So the System (ECA) can handle it. But if we observe Case 2, the user is Naive and does not

have knowledge about the broad and exact match. So the System has to explain about the services and then take the response from the user, where the dialog length is increased. If the system (ECA) can understand the user knowledge based on earlier conversation, it would be easy to ask the question along with the information about the subject of the question such that dialog length can be reduced. For Example, System response can be as follows:

System> *Would you like search to be a broad match or exact match? Where Broad match means …. And exact match means ….*

1.3.3 Insight into Existing Decision-Making Algorithms

Firstly, most existing decision-making algorithms like Breadth First Search (BFS), Depth First Search (DFS) and A* are limited to planning under definite environments and cannot handle the uncertainty in the environment. There are many algorithms proposed since 1960's to mathematically model decision-making under uncertainty and best among them are Markov Decision Processes (MDP). Since system (ECA) cannot observe the intention of the user directly, Partially Observable Markov Decision Processes (POMDP) which is an extension of MDP is more appropriate. Despite POMDP's undeniable advantages in handling uncertainty over other approaches, POMDP models cause significant loss of history information over interaction, leading to the untruthful recognition of user intention. In other words, POMDP helps to model till control level (decision making) but fails to maintain the user knowledge and addresses every user similarly, despite user's domain knowledge and intention. POMDP cannot handle problems mentioned in sections 1.3.1 and 1.3.2 because of loss of history information.

Various approaches [9] [33] [34] are proposed using POMDP and by storing the history information but still, there is a lot of scope to improve the intention discovery of the user.

Chapter 2 of this thesis comprehensively discusses drawbacks of POMDP and Shortcomings of existing approaches using history information. In summary, for the problems specified above i.e. for better user intention discovery and to optimize the dialog length thereby the automation of RE in SPL is effective:

a. Can we learn from the information space (history)?

b. Can we determine the user's knowledge level to address better?

1.4 A Novel Approach

The new approach aims to understand the requirement from the user as input by parsing it using natural language processing (NLP) and uses POMDP for decision making to find relevant services. For every decision made belief-state is computed and stored in POMDP. Trend analysis is performed on the belief-state history to anticipate the user knowledge level and to switch POMDP policies between different contextual control modes thereby improving the accuracy in intention discovery and also optimizing the dialog length.

1.5 The Thesis Statement

It is possible to effectively automate the Requirements Elicitation (RE) process in Software Product Line (SPL) using Embodied Conversational Agent (ECA) by performing trend analysis on belief-state history using Discrete Wavelet Transform.

By *effectively* automating, it is possible to improve the *accuracy* of user intention discovery and to *reduce* the dialog length over traditional POMDP model.

1.6 Summary of Thesis Contributions

In summary, the contributions of this thesis include:

- Effectively modeled the automation process of RE in SPL using the combination of POMDP, belief-state history and DWT.
- Analyzed the trend in belief-state history with DWT and associated the changing trend with the four modes of COCOM.
- Introduced different set of POMDP policies to recognize users with different knowledge levels thereby optimized the dialog length by performing appropriate actions in four modes of COCOM.

1.7 The Structure of This Thesis

The rest of the thesis is structured as follows: Chapter 2 provides an in-depth survey of existing works to automate RE in SPL, especially using POMDP, plus the concepts of trend analysis using the theory of wavelets and the use of it in automation. Chapter 3 discusses the proposed method of trend analysis on belief state history using DWT and to analyze the change in user intention thereby switching the contextual control mode to handle different user groups. Chapter 4 presents the experiment design of proposed method. The experiment design described in Chapter 4 is simulated on different user groups to obtain the results. Chapter 5 discusses the simulation methodology and results by comparing with the existing techniques to demonstrate the improved accuracy and efficiency. Finally, Chapter 6 discusses the conclusion and future work.

CHAPTER 2

A Literature Review

This Chapter discusses the decision-making algorithms, trend analysis methods, and previous works on information space and trend analysis thoroughly. End of the chapter summarizes the limitations of the existing algorithms, drawbacks of the previous works and also presents approaches to improve them.

2.1 Partially Observable Markov Decision Processes (POMDP)

POMDP is a generalization of a Markov decision process (MDP) and comes under probabilistic decision-making algorithms. Both POMDP and MDP are applicable for planning under uncertainty.

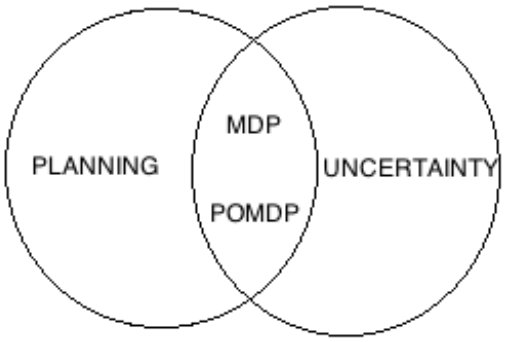


Figure 2.1: Decision-making algorithms lineage [14]

Planning algorithms are classified as follows:

Deterministic vs Stochastic - In deterministic environments, the outcomes of an action are predictable and always the same but in stochastic environments the outcomes of an action are random.

Fully Observable vs Partially Observable - When the states of an environment are visible then it is fully observable and when the states of the environment are partially visible then it is partially observable.

	DETERMINISTIC	**STOCHASTIC**
FULL OBSERVABLE	BFS, DFS and A*	MDP
PARTIALLY OBSERVABLE	-	POMDP

Table 2.1: Classification of decision-making algorithms

2.1.1 POMDP Model

A POMDP model [14] is a 7 tuple set $< S, A, O, T, \Omega, R, \gamma >$ where:

a. *States (S)* – The world is divided into a finite set of possible states. For example, possible states of a self-navigating robot are different positions (X and Y coordinates) it can take in a room.

b. *Actions (A)* – Finite set of possible actions available. Actions are Information driven or Goal driven.

c. *State Transition Function (T)* – It captures the probabilistic relationship between the states and the actions executed to change the state of the world.

$$T(s, a, s'): S \times A \times S = P(s' \mid s, a)$$

d. *Reward Function (R)* – It gives the relative measure of desirability to be in a state.

$$R(s, a): S \times A$$

e. *Observations (O)* – Finite set of observations of the state.

f. *Observation Function (Ω)* – It captures the probabilistic relationship between the state and observations.

$$\Omega (s, a, o): S \times A \times O = P (o \mid s, a)$$

g. *Discount Factor (γ)* – The discount factor decides how much immediate rewards are favored over future rewards. For example, when $\gamma = 0$ system chooses the largest immediate reward and when $\gamma = 1$ system chooses to maximize the expected sum of future rewards.

Using POMDP Model, the system makes decisions by executing the following steps:

Step 1: Use the policy to select an action for current belief state.

Step 2: Execute the action

Step 3: Receive an observation

Step 4: Update the belief state using current belief, action, and observation

Step 5: Repeat

2.1.2 Markov Property

Markov Property states that the optimal decision depends only on the current state [14]. It refers to the memoryless property of a stochastic process where there is no necessity to remember the history of what states were already visited, what actions were taken and what observations made.

In POMDP as the environment is partially observable, it is hard to say the decisions made are optimal as POMDP follows Markov property and does not remember history. It is also necessary to capture the relative likelihood of being in a particular state. POMDP uses the belief-

state concept in order to equip the factor replacing history. Markov Property is also referred as Markovian over belief-state.

2.1.3 What is Belief-State?

Belief-State is a probability distribution over all possible states which gives as much information as the entire action-observation history [1] [14]. In fact, belief-state along with transition and observation probabilities helps to transform the problem from partially-observable to completely observable. If we are given a belief state for time 't' and we perform an action 'a' and get observation o' we can compute a new belief state for time 't+1' by simply applying Bayes' Rule [14] and using the model parameters.

Belief-state Update:

$$b'(s') = P(s' \mid o', a, b)$$

$$= (P(o' \mid s', a, b)\, P(s' \mid a, b)) / P(o' \mid a, b)$$

$$= (P(o' \mid s', a) \Sigma_{s \varepsilon S} P(s' \mid a, b, s)\, P(s \mid a, b)) / P(o' \mid a, b)$$

$$= (P(o' \mid s', a) \Sigma_{s \varepsilon S} P(s' \mid a, b, s)\, b(s)) / P(o' \mid a, b)$$

Bayes Theorem (Conditional Probability):

$$P(A \mid B) = P(B \mid A)\, P(A) / P(B)$$

P(A) and P(B) – The probabilities of observing A and B exclusively

P (A | B) – The probabilities of observing event A given that B is true

P (B | A) – The probabilities of observing event B given that A is true

2.1.4 What are Policies?

In conventional planning, the system constructs a tree with states and actions. By traversing the tree from the root node to leaves, the optimal plan is computed. But in POMDP, as the outcomes of actions taken are stochastic in other words as the branching factor is high, the tree constructed using conventional planning is very deep. It is very hard to compute optimal plan by traversing the whole tree. To overcome these challenges, policies are introduced in POMDP. Policies are mapped from belief-states to actions.

Policy: belief-state → action

π: b(s') → a

2.1.5 Limitations using POMDP

Limitation of using POMDP for decision-making are as follows:

a. Because of Markovian property, POMDP model refrains to capture the history of actions taken and observations made which is a valuable information.

2.2 Trend Analysis

Trend Analysis is a technique for extracting an underlying pattern or behavior from the data. In general trend analysis is performed on historical data and time series data to predict the subject of interests for future. For example, trend analysis of rainfall for previous years helps us to determine the rainfall for the current year [1]. Different approaches of trend analysis are as follows [7]:

a. *Sampling* – In Sampling, the historical data is split into training and testing datasets. The training dataset is used to develop a predictor model and its accuracy is

determined using the testing dataset. Random Sampling and Reservoir based sampling are few sampling methods.

b. *Histogram* – Trend is analyzed by constructing a histogram from the historical data by dividing the entire range of values into a series of intervals and then count how many values fall in each interval. Equi-Depth and V-Optimal are few Histogram approaches.

c. *Sketches* – The frequency distribution of historical data is summarized by using hash functions. Count Sketches and Count-Min Sketches are few examples.

d. *Wavelets* – Mathematical transformations are applied to transform the data into a set of wavelet coefficients representing a different level of granularity to analyze the trend.

This thesis primarily focuses on using the theory of wavelets for trend analysis as wavelets outperform other trend analysis approaches for time-series data [4] [7]. Later sections of this chapters discuss Signals, Frequencies, and Transformations (FT, STFT) as they form a necessary background to understand how the Wavelet Theory (WT) works.

2.2.1 Signals, Frequencies and Transformations

What is signal?

Signal is the time-amplitude representation of a graph plotted between time (independent variable) on x-axis and amplitude on the y-axis (dependent variable) [23]. In most cases, distinguishable information is present in the frequency components of a signal.

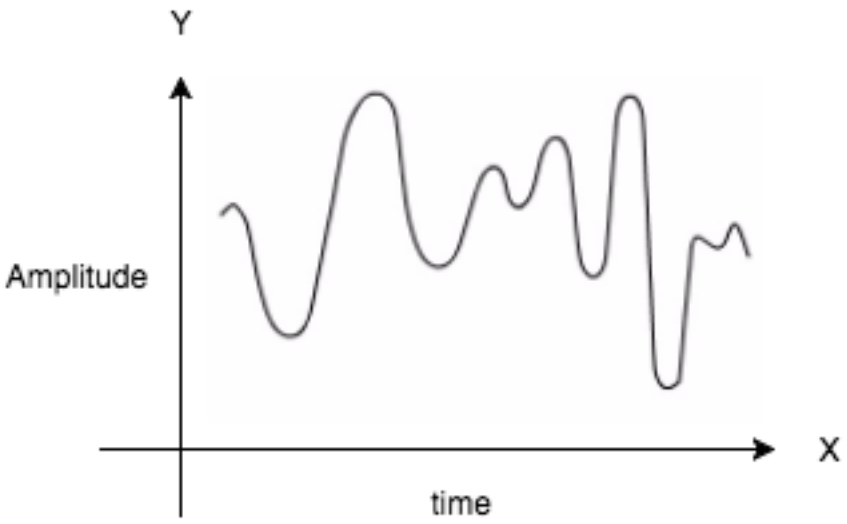


Figure 2.2: Sample Signal

What is frequency?

Frequency is the rate of change of a variable. If the variable changes rapidly, it is high frequency. If the variable change is smooth, it is low frequency. Mathematical transformations are applied to signals to obtain further information (frequency spectrum) which is not readily available in raw signal. For example, on applying mathematical transformation to the unknown signal in Figure 2.3, we understand how much of each frequency exists in the unknown signal. The plot in Figure 2.4 tells that it consists of following frequencies: 10, 25, 50 and 100 HZs.

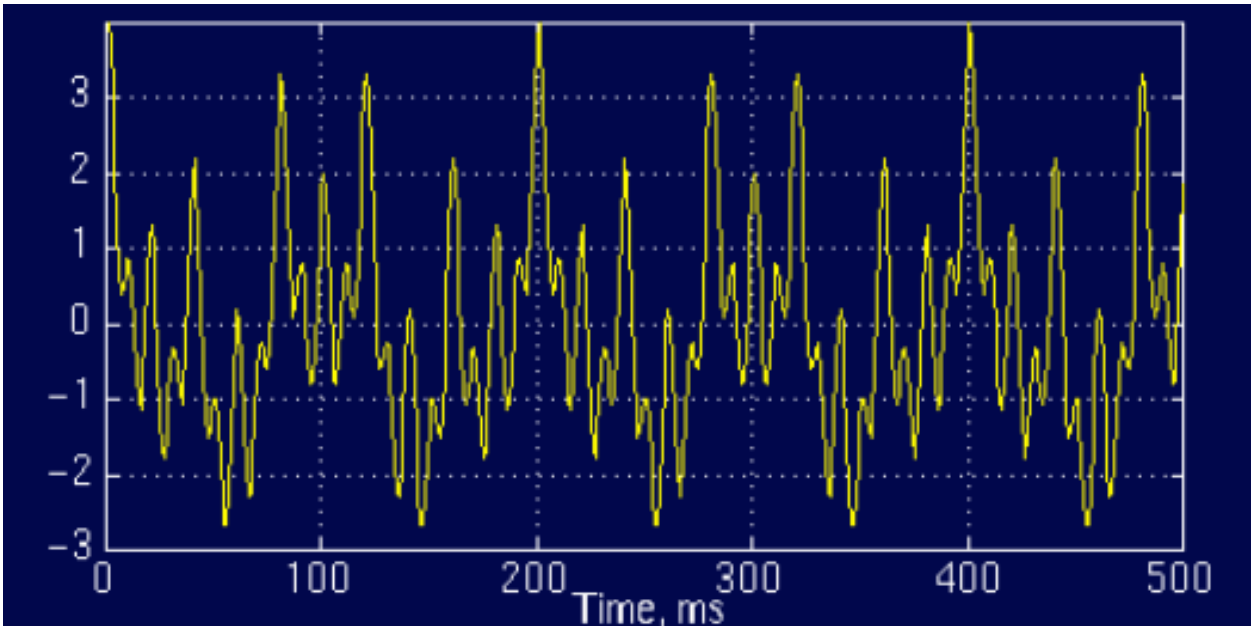


Figure 2.3: Unknown Stationary Signal [23]

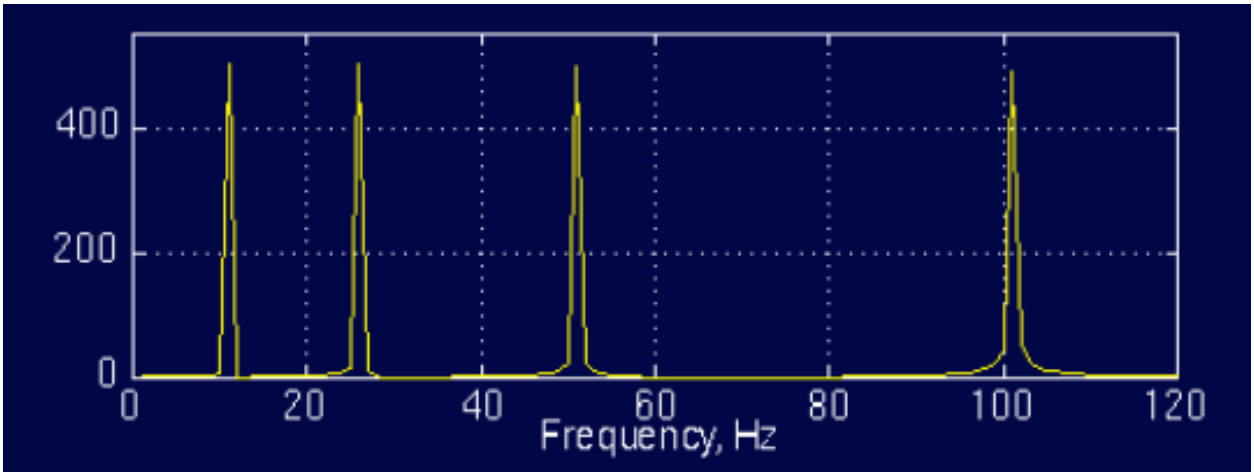


Figure 2.4: Frequencies in Unknown Stationary Signal illustrated in Figure 2.3 on applying FT [23]

Stationary Vs Non-Stationary Signals?

For Stationary Signals, all the frequency components that exist in the signal, exist throughout the entire duration of the signal. For example, the signal in Figure 2.3 is a stationary signal and it has same frequencies of 10, 25, 50 and 100 HZ at all times. For non-stationary signals, the frequency

components change over time. For example, the signal in Figure 2.5 is a non-stationary signal where the frequency of 10HZ exists from 0 to 300 ms only.

2.2.2 Fourier Transformation (FT)

Fourier Transform (FT) transforms a signal to complex exponential functions of different frequencies and represented from following equations [23]:

$$X(f) = \int_{-\infty}^{\infty} x(t) \cdot e^{-2j\pi ft}\, dt$$

$$x(t) = \int_{-\infty}^{\infty} X(f) \cdot e^{2j\pi ft}\, df$$

In the above equations, f stands for frequency and t stands for time. $x(t)$ denotes the original signal in time domain and $X(f)$ denotes the signal in frequency domain. The exponential term can be replaced with combination of cosines and sines as follows:

$$e^{-2j\pi ft} = \cos(2\pi ft) + j\sin(2\pi ft)$$

In other words, the raw/original signal is multiplied with above equation over all times from $-\infty$ to ∞. The result of the integration is a large value, then the raw/original signal $x(t)$ has a frequency component. The result from the FT is a plot with frequency on X-axis and amplitude on Y-axis. It misses capturing the details of what frequencies exist at what times. FT is not suitable for non-stationary signals. For example, the resultant frequencies for signals in Figure 2.3 (stationary) and Figure 2.5 (non-stationary) are same as shown in Figure 2.4 and Figure 2.6 respectively.

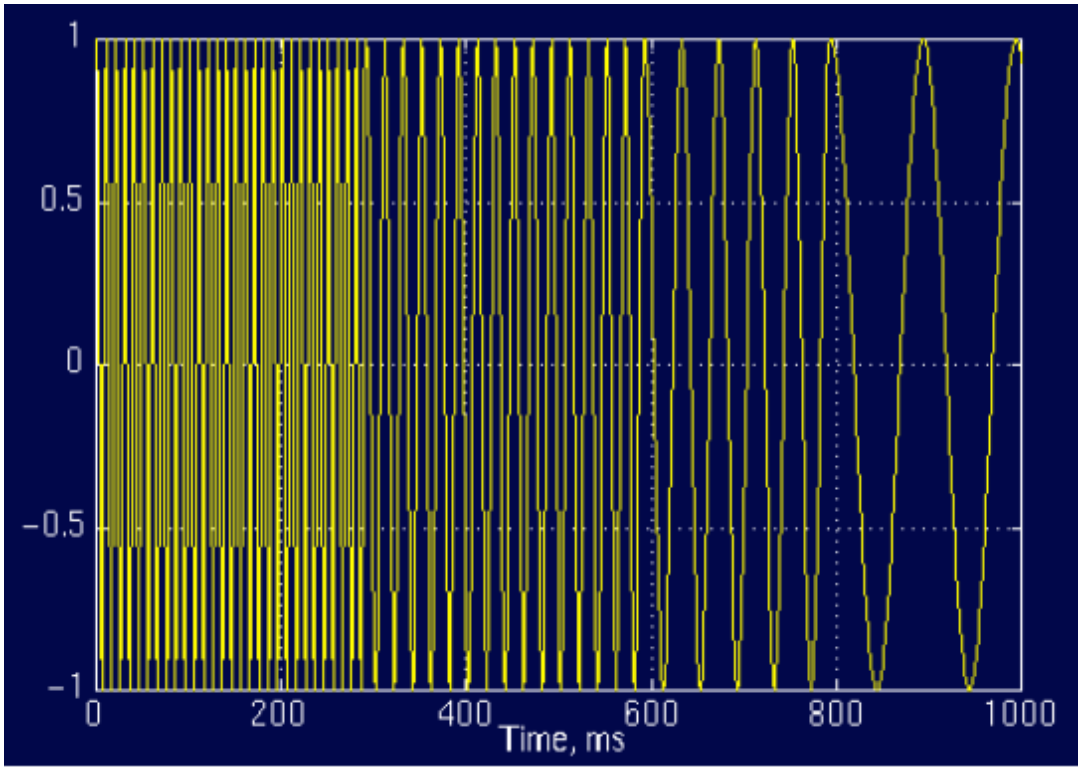


Figure 2.5: A non-stationary signal [23]

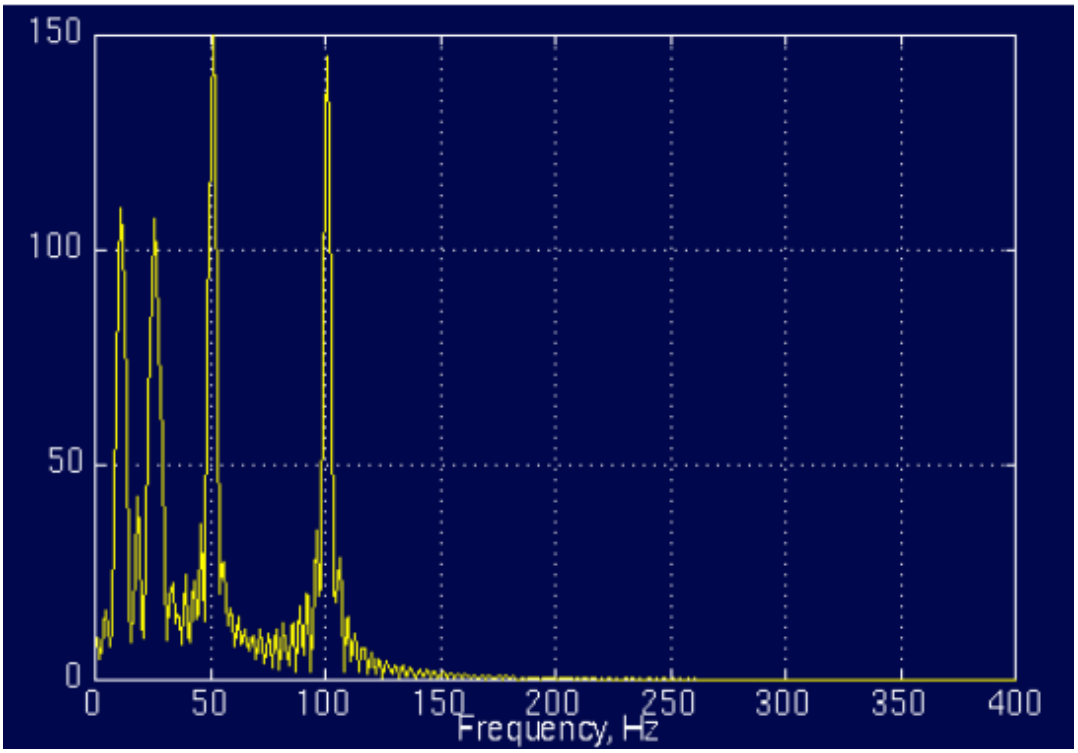


Figure 2.6: Frequencies in non-stationary signal illustrated in Figure 2.5 on applying FT [23]

2.2.3 Short-Time Fourier Transformation (STFT)

STFT is a revised version of FT which gives the time-frequency representation (TFR) of the raw/original signal. Based on the idea that a non-stationary signal composes of stationary signals for fixed duration. In STFT the signal is divided into small segments by applying window function where each segment is a valid stationary signal.

$$STFT_X^W(t', f) = \int_t [x(t) \cdot W(t - t')] \cdot e^{-j2\pi f t} dt$$

In the above equation t' is window interval, t is finite time and f is the frequency. $x(t)$ is the raw/original signal and $W(t)$ is a window function.

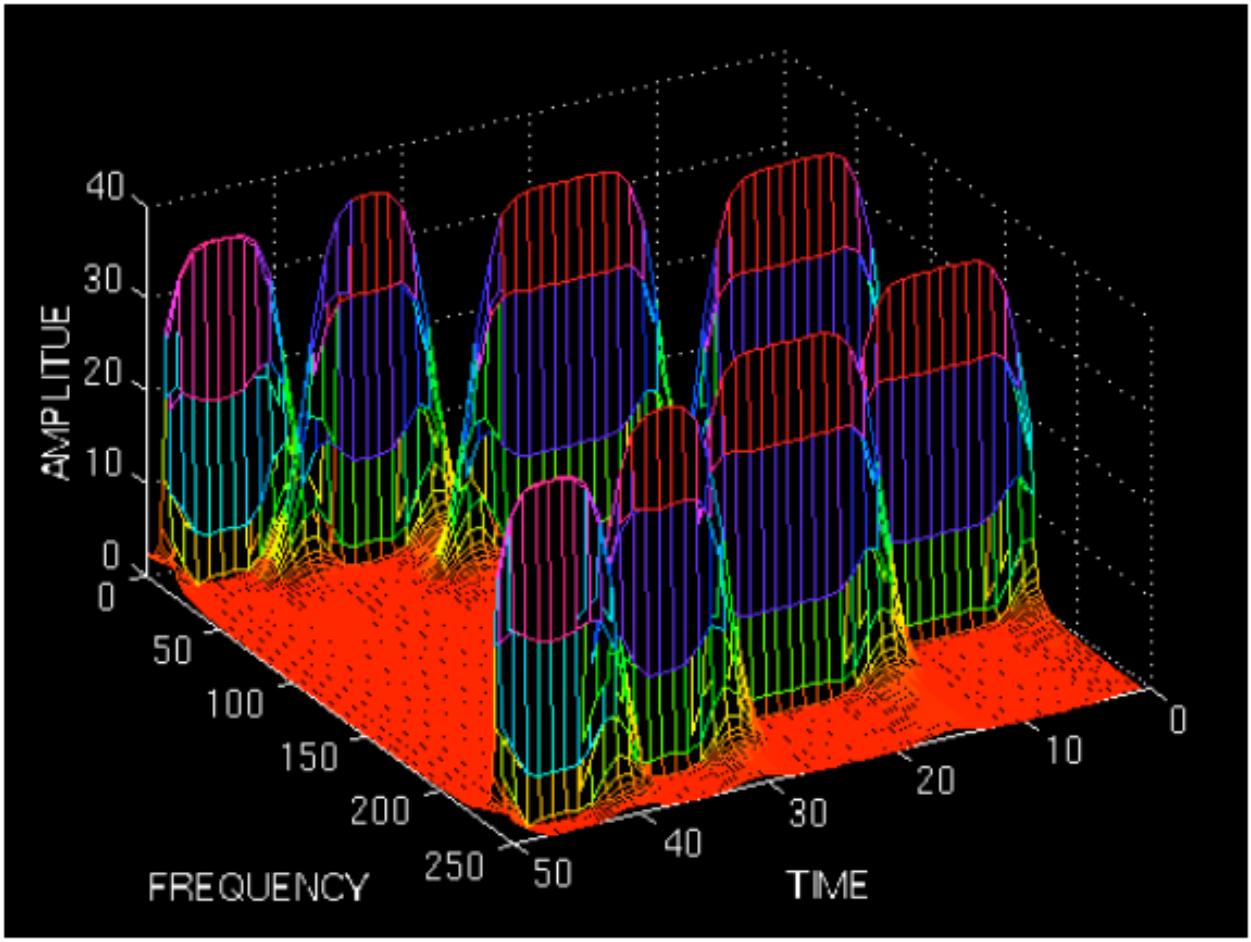


Figure 2.7: FTR for non-stationary signal illustrated in Figure 2.5 using STFT [23]

From the FTR as illustrated in above diagram, STFT provides details about what frequencies exist at what time. But the problem with STFT is the fact whose sources go back to Heisenberg Uncertainty Principle [23]. The principle says that the frequency and time of a signal cannot be known simultaneously. STFT provides the time intervals in which certain band of frequencies exists, which is a resolution problem. The reason for having time and frequency resolution problems is window size. The narrow window gives good time resolution and poor frequency resolution where wide window gives good frequency resolution and poor time resolution.

2.2.4 Theory of Wavelets (WT)

Wavelet means a small wave. The term 'small' refers that the function of finite length and the term 'wave' refers that the function is oscillatory. The transformation function in wavelet theory is mother wavelet. The term mother implies the one main function which generates the transformation functions needed for the whole transformation process. Examples of mother wavelet are Morlet wavelet and Mexican hat function [24].

2.2.4.1 Continuous Wavelet Transformation (CWT)

The CWT was developed as an alternative approach to STFT to address the time and frequency resolutions.

$$CWT_x^{\psi}(\tau, s) = \psi_x^{\psi}(\tau, s) = \int x(t) \cdot \psi_{\tau,s}^{*}(t)\, dt$$

$$\psi_{\tau,s} = \frac{1}{\sqrt{|s|}} \psi\left(\frac{t-\tau}{s}\right)$$

From the above equation [23], the transformed signal is a function of two variables, τ and s, the *translation* and *scale* parameters respectively. $\psi^*_{\tau,s}(t)$ is the transforming function i.e. the mother wavelet.

What is translation?

The term translation refers to the location of window in terms of time, as the window is shifted through the signal.

What is scale?

Scale is the inverse of frequency and as a mathematical operation it either dilates or compresses a signal. High scales (low frequencies) gives the global information of a signal, whereas low scales (high frequencies) corresponds to a detailed information of hidden pattern in the signal. From the definition of WT, as the scale parameter is defined in the denominator, scales $s > 1$ dilates the signal and $s < 1$ compresses the signal.

Often the signals encountered in the practical application have high-frequency components for a short duration and low-frequency components for a long duration. WT is designed to give good frequency resolution and poor time resolution at low frequencies and good time resolution and poor frequency resolution at high frequencies.

How the computation of CWT works?

The mother wavelet serves as a prototype for the wavelet analysis. Two commonly used mother wavelets are Morlet wavelet and Mexican hat function [24]. The Morlet wavelet is defined as follows:

$$W(t) = e^{iat} \cdot e^{\frac{-t^2}{2\sigma}}$$

where a is the modulation parameter and $\sigma(sigma)$ is the scaling parameter which affects the width of the window. The Mexican hat function is defined as follows [24]:

$$\psi(t) = \frac{1}{\sqrt{2\pi}\sigma^3}\left(e^{\frac{-t^2}{2\sigma^2}} \cdot \left(\frac{t^2}{\sigma^2} - 1\right)\right)$$

where $\sigma(sigma)$ is the scaling parameter. Once the mother wavelet is selected, the computation starts with scaling parameter as $s = 1$ and CWT is computed for all values of s (smaller and larger than 1) by translating (shifting) the signal by τ(tau) each time. For real world applications, it is not necessary to compute the CWT for all values of s and τ(tau) and can be limited.

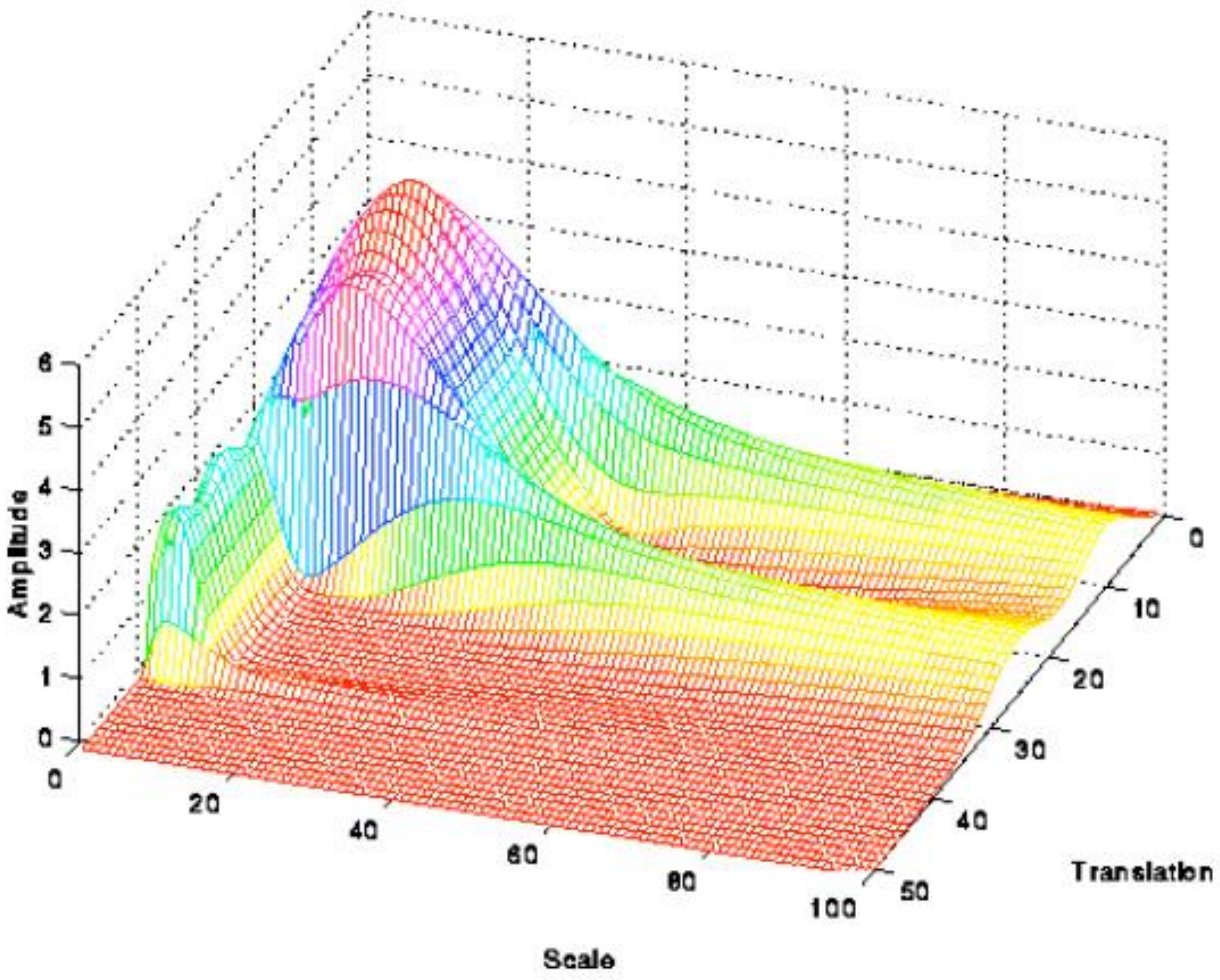


Figure 2.8: FTR for non-stationary signal illustrated in Figure 2.5 using CWT [23]

Note that in Figure 2.8, CWT has a good time and frequency resolutions at high and low frequencies respectively rather than constant resolution using STFT (refer Figure 2.7).

The computation of FT or STFT or CWT by using analytical equations and integrals is practically very hard for computers [23]. Particularly, CWT provides highly redundant information which requires a significant amount of computation time and resources. Therefore it is necessary to discretize the transforms.

2.2.4.2 Discrete Wavelet Transformation (DWT)

The DWT decompose signal to discrete time and provides sufficient information for both analysis and synthesis of the original signal within signification computation time. The discretization of the signal is achieved by subsampling and upsampling operations. Subsampling refers to the removal of some of the samples of the signal by reducing the sampling rate. Upsampling refers to the addition of new samples to the signal by increasing the sampling rate. In DWT, coefficients are obtained by sampling with $s = 2^j$ and $\tau = k * 2^j$. DWT analyzes signal by decomposing signal at different frequencies to address different resolutions. DWT uses two sets of functions called scaling and wavelet functions. The original signal x[n] is first passed to high-pass filter g[n] and low pass filter h[n] iteratively to obtain the frequencies.

$$y_{high}[k] = \sum_n x[n] \cdot g[2k - n]$$

$$y_{low}[k] = \sum_n x[n] \cdot h[2k - n]$$

where $y_{high}[k]$ and $y_{low}[k]$ are the outputs of high-pass and low-pass filters. For every iteration the signal is subsampled by 2 to obtain the remaining frequencies. The decomposition of the original signal and coefficients obtained at each level is illustrated in Figure 2.9.

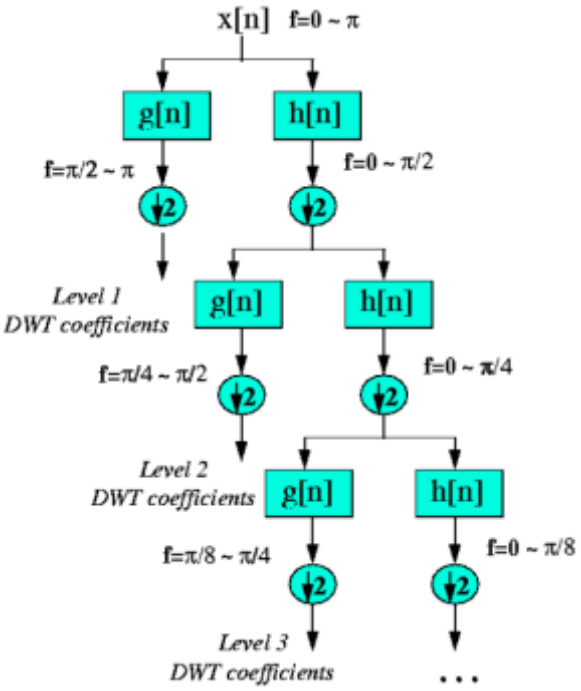


Figure 2.9: Decomposition of signal using DWT [23]

The coefficients obtained at different levels are plotted on graph to understand the frequency content within the original signal. In this thesis, the number of sharp variation points obtained from graph (b) in Figure 2.10 are used.

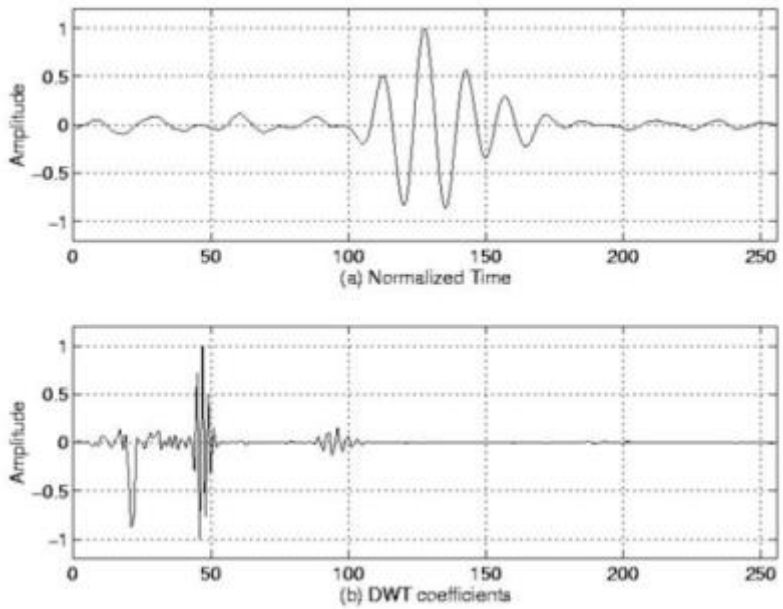


Figure 2.10: Example of DWT [23]

2.3 Contextual Control Models (COCOM)

Humans can achieve their goals by doing many things in many different ways and the performance is mainly determined by the situation. In other words, the selection among the possible actions is not determined by the characteristics of action elements but by the current needs and constraints. A COCOM is based on three main concepts: competence, control and constructs. The COCOM simplifies the description of control through four control modes [30]:

a. Scrambled – In Scrambled control mode, the next action is random or irrational. There is a little or no correspondence between the situation and the actions. In other words, it is an extreme situation with zero control.

b. Opportunistic – In Opportunistic control mode, the actions made are heuristic due to lack of clarity about current situation. Planning and anticipation are limited in this mode.

c. Tactical – In Tactical control mode, the next action is more or less follows a known procedure or rule. Planning for limit scope is considered where more focus is on fulfilling the dominant needs of the present.

d. Strategic – In Strategic control mode, the choice of actions has very less influence on the situation. The interaction between multiple goals can also be taken into account for planning in this mode.

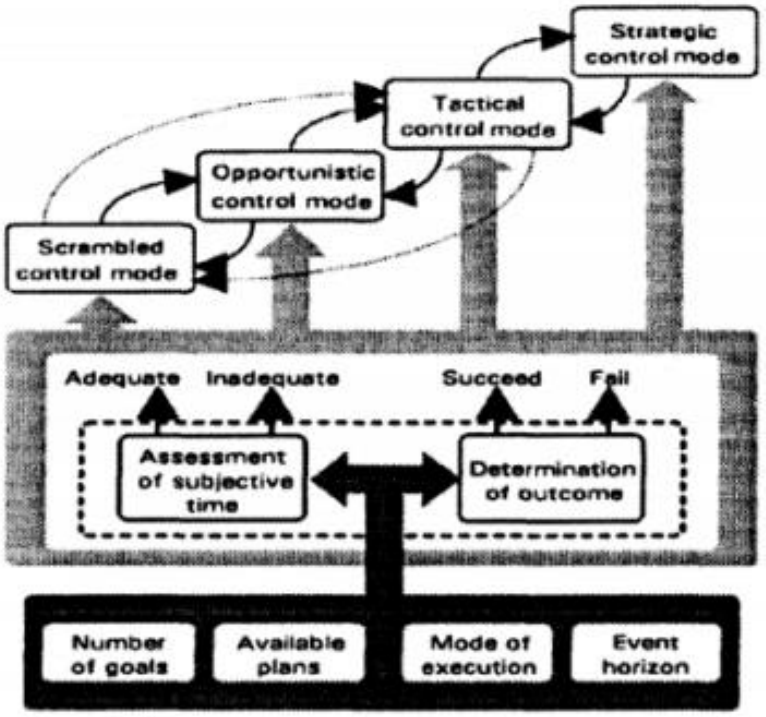


Figure 2.11: Internal Structure of COCOM [30]

2.4 Previous Works

The primary focus of this thesis is to automate the requirement elicitation in SPL. In other words, automation of requirement elicitation means the software is customized to meet the needs of user requirements. Various approaches [9] [19] [27] [34] [35] are published to interactively gather the requirements from the user for software customization using Dialog Interfaces, Software Visualization techniques, and Conversational Web Agents. Subsequent sections of this chapter discuss the previous works, summarizes their contributions and limitations.

2.4.1 An Interactive approach by using Dialog Interface

Zhang et al. [35] proposed a dialog-based interactive approach for software customization using ontology-based requirement elicitation. The framework designed is text-based and has four components: Dialogue Interface, I/O Controller, Dialogue Manager and Ontology Knowledge Base as illustrated in the Figure 2.10.

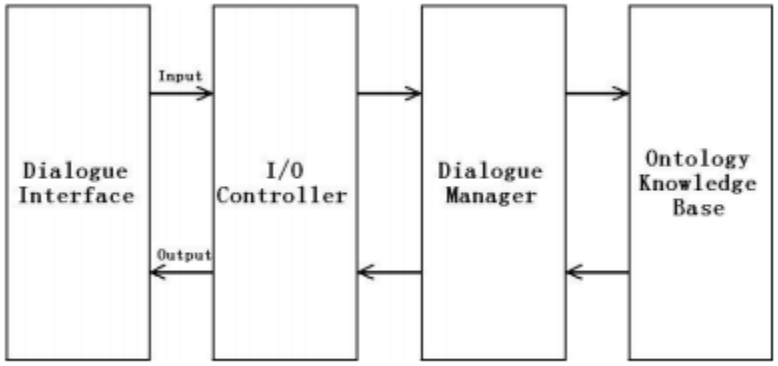


Figure 2.12: The ontology-based requirement elicitation model [35]

The Dialogue Interface displays machine generated text from I/O controller and also has a slot for the user to fill decision about the requirement as YES or NO. If the answer matches with predefined information, I/O controller passes the user's input to Dialog manager. The Dialog Manager evaluates the requirement after receiving the decision from the user and it consults with the ontology knowledge base for software customization. I/O controller receives the output from Dialog Manager. It translates output into natural language and displays the message on Dialog Interface. The process is repeated until the user's goal is reached. The evaluation of requirements in Dialog Manager is facilitated through following functions: Generalize, Decompose, Rely, Contradict, Associate, hasRank, and Invalid.

2.4.2 An Interactive approach by using Software Visualization

Sadri et al. [27] have enhanced the approach in [35] by adding a Petri-Net based method of graphical visualization for software customization. It aims primarily to enhance the user understanding and reduces the time, cost and effort spent on the previous system thereby increasing the overall usability of the system.

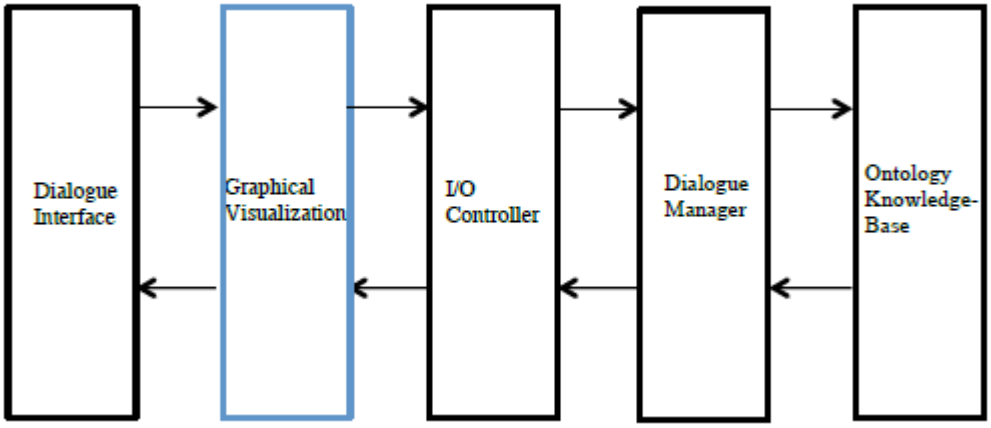


Figure 2.13: A Petri-Net based method of graphical visualization for software customization [27]

The author discusses different visualization techniques like Flow Chart, State Diagram, Activity Diagram, and Petri Nets. Petri Nets is chosen as its more suitable for demonstrating the workflow behavior of the system. Petri-net is a special type of directed graph with two types of nodes called places and transition, which are illustrated by circles and rectangles respectively. An arc will connect each place to a transition and each transition to a place. The contributions are justified through usability study by comparing the text based approach in [35] with the graphical interface illustrated in Figure 2.13.

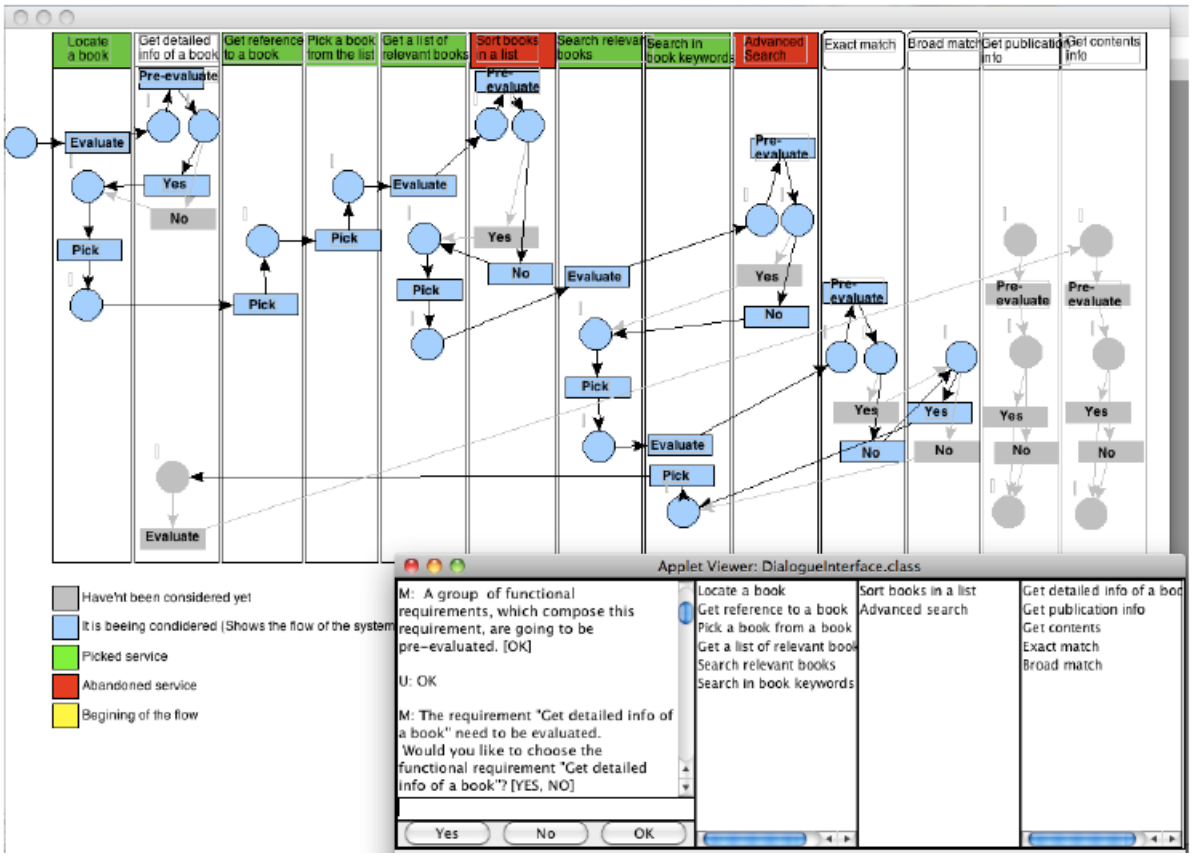


Figure 2.14: The enhanced text-based system with the graphical interface [27]

Kaler et al. [19] have improved the approach in [27] by adding an interactive visualization to improve the usability of the system. In this approach, COCOM is used to classify users into four groups based on the knowledge level of the software customization and software development. These users are provided with different visualization interfaces as per their knowledge. The four different visualizations approaches are as follows: Petri nets based, directed graph based, requirement model based and block based visualization. A usability study is conducted and evaluated that the aimed interactive visualization enhances the learnability for real users.

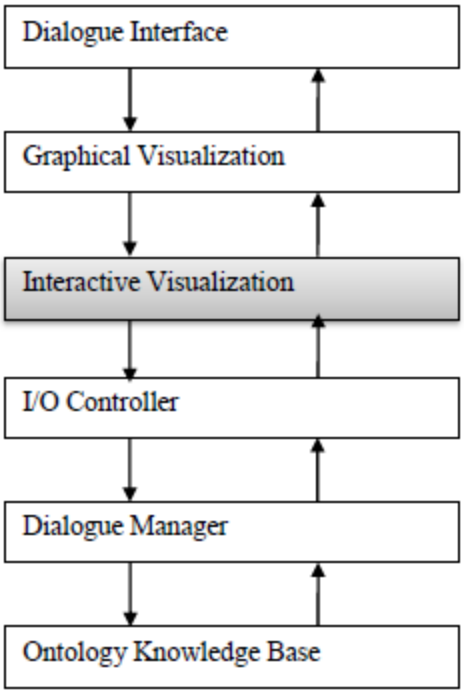


Figure 2.15: An Interactive visualization for software customization [19]

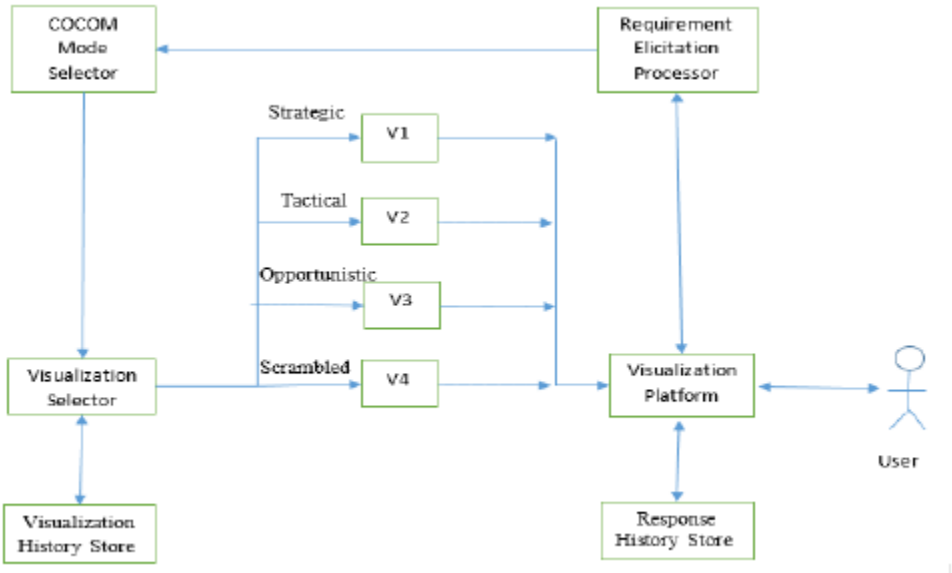


Figure 2.16: COCOM based interactive visualization architecture [19]

2.4.3 An Interactive approach by using Embodied Conversational Agents

Dhanapal et al. [9] has proposed a model by introducing belief state history into POMDP based dialog management system. The belief state history is analyzed and rate of change of trend in belief state is observed. The next belief state is predicted using data mining techniques such as Apriori algorithm to switch between different contextual modes for better user intention discovery. The proposed approach is evaluated by experimenting under different scenarios.

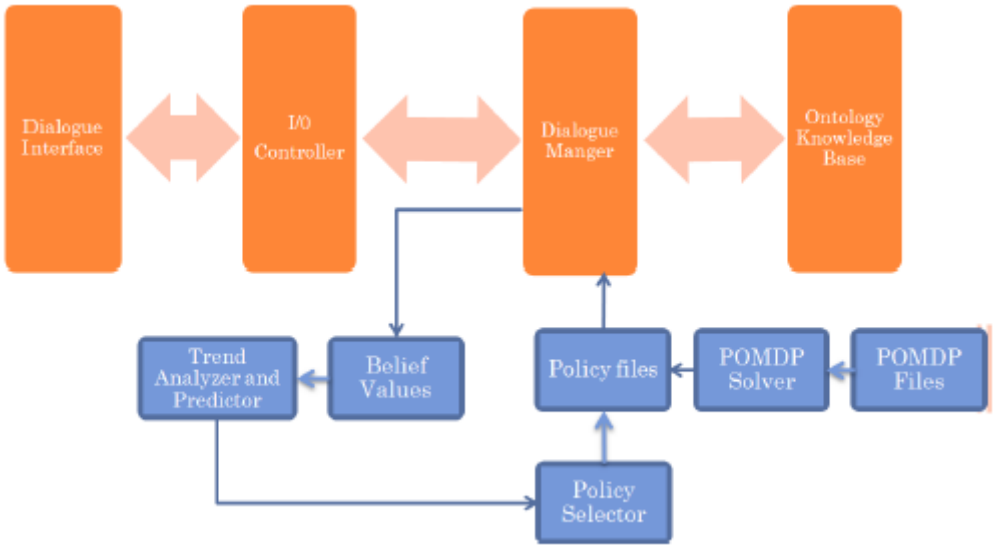


Figure 2.17: Contextual Control in Dialogue Management with Belief State Prediction [9]

Dialog Manager receives observation in natural language as YES/NO from the Dialog interface through I/O controller. The belief state values are computed in Dialog Manager and trend analysis is performed on history of belief states to predict the next belief state (Trend Analyzed and Predictor). Policy selector selects the policies based on the predicted next belief state value by referring the Policy files which consists of the states, action, rewards and policies predefined.

2.4.4 Affective Dialogue Modelling using POMDP

Bui et al. [2] [3] proposed an approach to develop a dialogue model which is able to take emotional state aspects of user into account and can act accordingly. Their model is based on POMDP which observes the observations composed of emotional state and action. Therefore, to create an affective dialog manager process, two inputs were used. One is emotional state and another is actions of the user. As the two inputs emotional state and actions are very uncertain and can change quickly so, their model is based on POMDP which is suitable for designing affective dialog models.

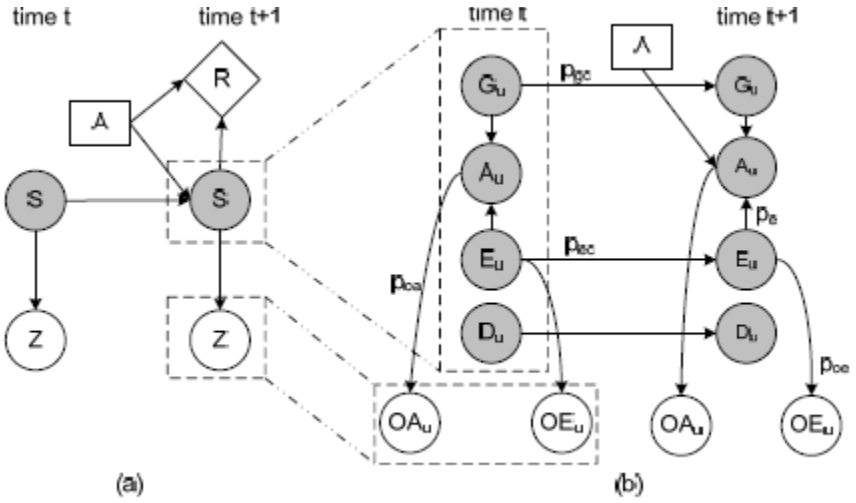


Figure 2.18: (a) Standard POMDP, (b) Two time-slice of factored POMDP for the ADM [2] [3]

The state set and observation set are composed of six features. As shown in Figure 1a the state set is composed of the user's goal (G_u), the user's emotional state (E_u), the user's action (A_u), and the user's dialogue state (D_u). In the observation set, the (OA_u) is observed user's action and the (OE_u) is observed user's emotional state. Figure 1b shows the affective dialogue model (ADM). The features of the state set, action set, observation set, and their correlations form a two time-slice Dynamic Bayesian Network (2TBN) which is built for the "route navigation in an unsafe tunnel" example used for experimental work. 2TBN can be easily modified to represent

other correlations, for example the correlation between the user's goal and emotional state. The 2TBN representation is allowing integrating the features of states, actions, and observations in a flexible way. The author states that if the observation is perfect, the expected return of the optimal dialogue strategy depends on the correlation between the user's emotion state and the user's action. The model is still lacking in scaling up the model with larger state, action, and observation sets for real-world dialogue management problems, extending the model representation, collecting and generating both real and artificial data to build and train the model.

2.5 Summary

In summary, the limitations of existing decision-making algorithms, trend analysis approaches and drawback of previous works to automate the RE in SPL are as follows:

- POMDP model refrains to capture the history of actions taken and observations made. This history information can be used as a knowledge base to discover interesting facts about the environment (user).
- Sampling, Histogram, and Sketches are statistical-based approaches and are inadequate to predict the hidden patterns when applied to time-series data.
- Fourier Transform (FT) is suitable to understand what frequencies exists in a signal but misses to capture at what times these frequencies occur.
- Short Time Fourier Transform (STFT) provides the TFR for an signal but has frequency and time resolutions.
- Continuous Wavelet Transform (CWT) is hard to compute in practical application and discretization is necessary.

- Approach [35] using Dialog Interface lacks Interactivity, Natural Language processing of user input (not just Yes/No text), user intention discovery for better software customization and optimizing dialog length.
- Approaches [19] [27] using Visualization techniques provides a flavor of interaction but still lacks Natural Language processing of user input (not just Yes/No text), user intention discovery for better software customization and optimizing dialog length.
- Approach [9] using ECA provides better user interaction than [19] [27] but still lacks Natural Language processing of user input. It addresses to improve the user intention discovery using naive data mining techniques which captures the hidden patterns but misses to identify frequency aspects in time-series data.
- Approaches [2] [3] effectively models the dialog but misses to capture the history and learn from it.

CHAPTER 3

THE PROPOSED METHOD

3.1 Overview

This Chapter discusses the architecture of the proposed method, steps involved by flow diagram, modified POMDP Model and algorithms developed thoroughly.

3.2 Architecture

The user interacts with the agent by providing the *observation* and agent responds to the user by performing an *action*.

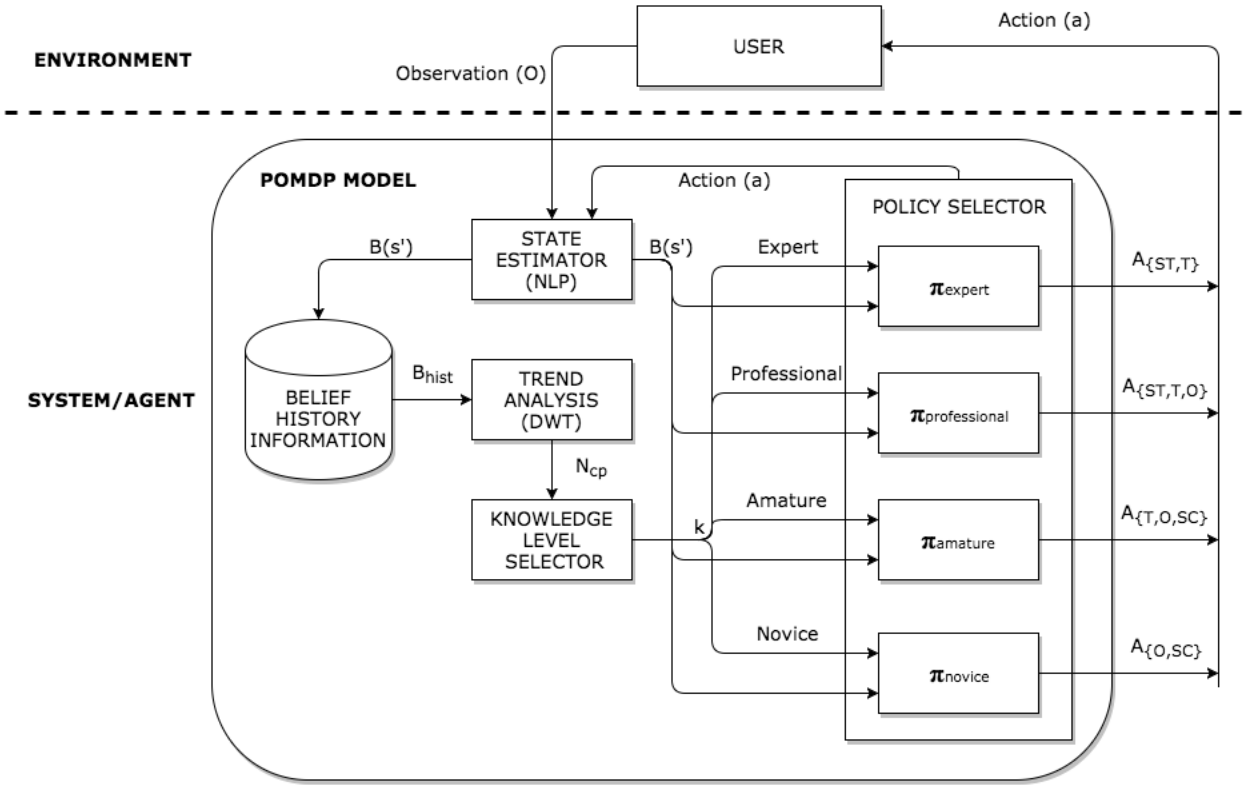


Figure 3.1: Architecture Diagram of proposed framework

Architecture diagram in Figure 3.1, illustrates the different modules enclosed within the agent. Later sections (refer 3.4) of this chapter explain the algorithms involved in each module. The modules are as follows:

a. *State Estimator (SE)* – It receives the observation as input from user. SE computes the observation probabilities through NLP and updates the belief-state value $B(s')$.

b. *Belief History Information Storage* – The belief-state $B(s')$ value computed in the SE is stored in the belief history information (b_{hist}) storage module.

c. *Trend Analysis* – It receives the history of belief information (b_{hist}) as input. Number of sharp variation points (N_{cp}) is obtained by performing DWT on b_{hist}.

d. *Knowledge Level Selector* – It receives sharp variation points (N_{cp}) as input and decides the knowledge level (k) of the user based on the knowledge level thresholds obtained from training (refer 3.5).

e. *Policy Selector* – It receives knowledge level (k) of the user and belief state $B(s')$ value as input. In Policy Selector, different set of policies are defined for users at different knowledge level. Policy (π_k) is selected based on the value of $B(s')$.

f. *Make Action* – It is a within the Policy Selector module which receives the policy (π_k) to execute. Each policy in POMDP is mapped from $B(s')$ to actions. In Make Action module, action (a) is performed through COCOM modes.

The sequence of computations that are happening for every interaction between user and agent is illustrated thoroughly as Flow chart in Figure 3.2. Agent will prompt user to check if user has achieved goal to end the requirement elicitation process by automatically generating customized software with selected services.

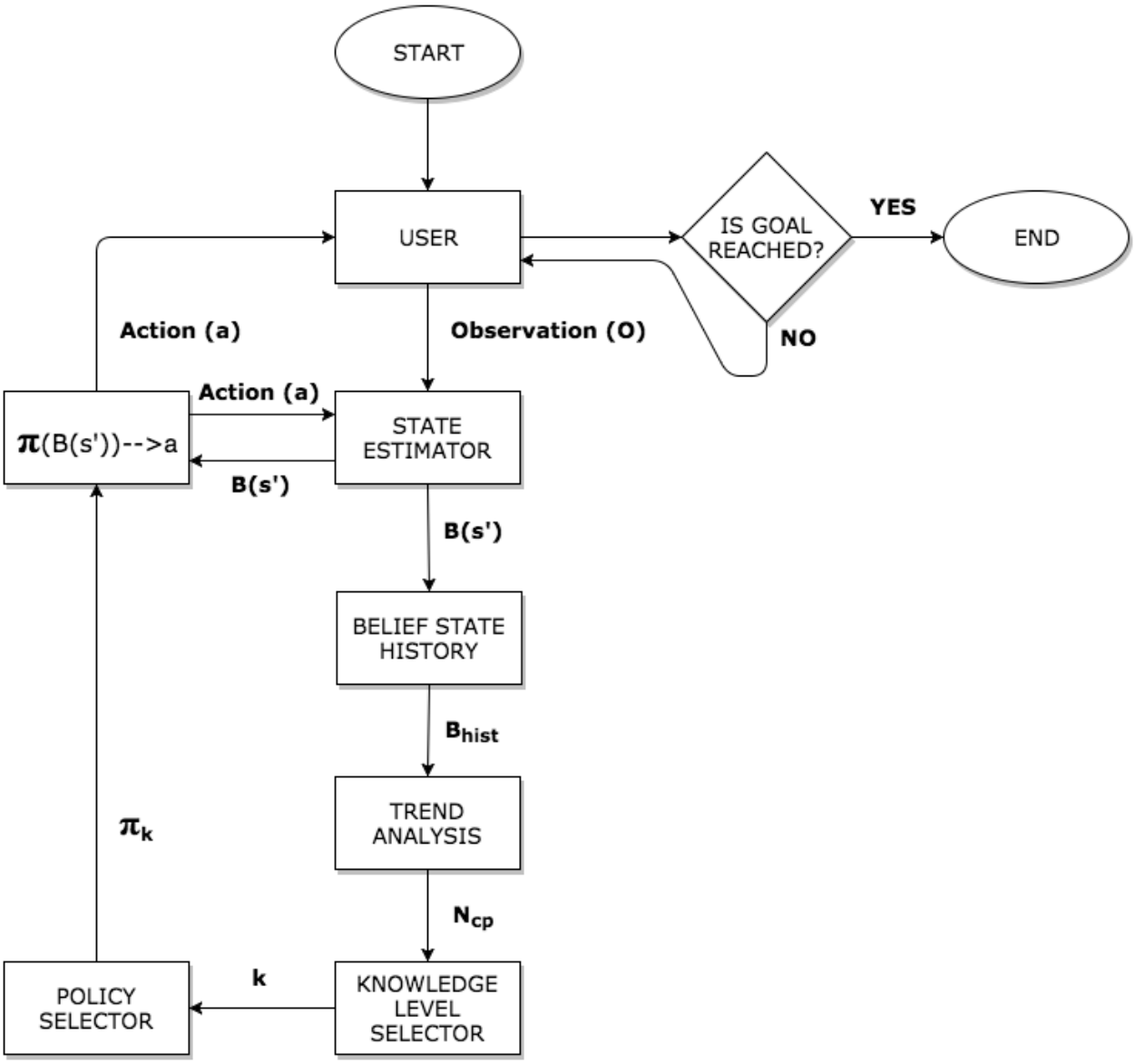


Figure 3.2: Flow Chart for every interaction between user and agent

3.3 Modelling POMDP

This section discusses 7 tuples $< S, A, O, T, \Omega, R, \gamma >$ in POMDP model such as States (S), Actions (A), Observations (O), Transition (T) and Observation Probabilities (Ω), Reward Functions (R) and Discount Factor (γ) are defined as follows:

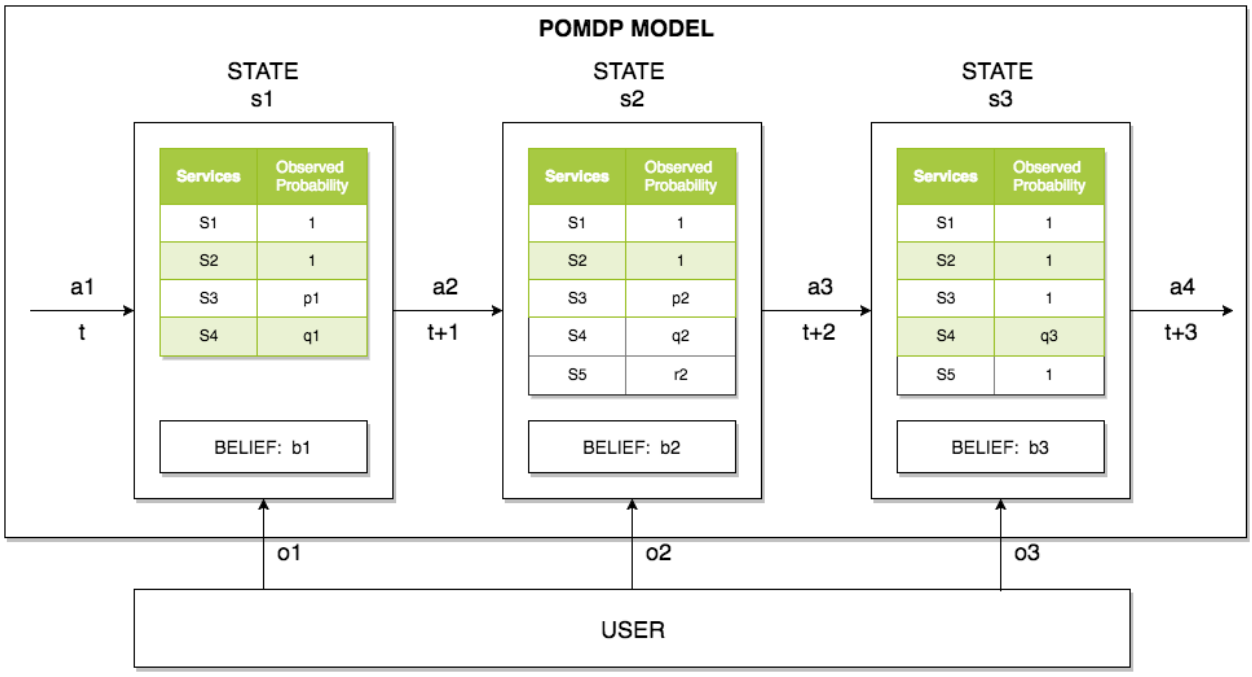


Figure 3.3: POMDP Model

3.3.1 States (S)

The primary focus of this thesis is to automate the RE in SPL thereby producing the customized software as per the requirements of the user. So, the list of services recognized from the requirements given by the user is the world for the agent.

As illustrated in Figure 3.3, the states of the proposed POMDP model are defined as the list of services with their respective observation probabilities. The State $s1$ at time t holds the services $S1, S2, S3, S4$ with observation probabilities $1, 1, p1, q1$ respectively. Where the observation probability of 1 means that the services $(S1, S2)$ are already selected by the user.

3.3.2 Actions (A)

Actions in proposed POMDP are either Information gathering actions or goal driven actions. Sample actions are as follows:

a. Requested service/feature is selected. // Strategic Mode

b. *Sorry, which of the following do you want: broad match search or exact match search? // Scrambled Mode*

c. *Did you mean: exact match search? // Tactical Mode*

d. *Sorry! Can you explain about it more? // Opportunistic Mode*

As illustrated in Figure 3.3, $a1, a2, a3, a4$ are the actions performed. As the actions are information gathering or goal driven, action $a2$ can increase or decrease the probability of services $S3$ and $S4$ from $p1\ to\ p2$ and $q1\ to\ q2$ respectively. Also action $a2$ can also gather information about service $S5$ and add it to the list in State $s2$. In proposed POMDP model, actions are the *strings* derived from different contextual modes.

3.3.3 Observations (O)

Observations are inputs from user to the agent in proposed POMDP model. For example, observation can be a requirement or selection of a service or requesting information about a service. Sample observations are as follows:

a. *I need search by keywords feature for my book store // requirement*

b. *Yes, I will go with broad match search feature // selection*

c. *Can you explain what is exact search means? // requesting information*

The observation received from the user is analyzed using NLP to understand the intention of the user and belief-state value is computed as described in Chapter 2 using Bayes theorem.

3.3.4 Transition (T) and Observation Probabilities (Ω)

In the proposed POMDP, the states are dynamic and dependent on the observations that are received throughout the interaction. Observations are in turn partially dependent on Actions.

As illustrated in the Figure 3.3, there can infinitely possible states agent can make transaction to for a given action. For example, based on the observation $o1$ received from user, agent can make a transition from state $s1$ to unknown infinitely possible state where there could be new services that added to the list, services can be removed, services can improve/decrease the observation probabilities. So, in proposed POMDP the transition probabilities (T) are assumed to be having equal probability.

Observation probabilities (Ω) are computed by tokenizing the input string given by the user. Using NLP, semantics and string similarity functions comparing input string with the service description Ω for different services can be calculated. NLP, sematic and string similarity can be easily achieved by using pre-built popular libraries mentioned software section (refer 4.2) in Chapter 4.

3.3.5 Reward Function (R) and Discount Factor (γ)

In General, reward functions are defined for the actions made from various status. In proposed POMDP model, since the strategy for actions made are based on contextual control modes. The rewards are randomly assigned based on logic reasoning that Strategic mode is better than Scrambled mode:

a. Reward(State, Strategic Mode) → 100

b. Reward(State, Tactical Mode) → 50

c. Reward(State, Opportunistic Mode) → 0

d. Reward(State, Scrambled Mode) → -50

Discount Factor (γ) is assumed as 1 to maximize the future sum of rewards. Refer section 2.1.1 for more detailed explanation.

3.3.6 Deriving Formula for Updating Belief-State

The key for POMDP's decision making is in updating the belief-state for every action and observation received. To update the belief-state, the below equation is taken from the section 2.1.3 where a is the action, o' is the observation received from transition from state s to s'.

$$b'(s') = (P(o' \mid s', a) \Sigma_{s \varepsilon S} P(s' \mid a, b, s) b(s)) / P(o' \mid a, b)$$

Since the transition functions are assumed to have equal probability distribution, the above equation will transform into following:

$$b'(s') = P(o' \mid s', a) / \Sigma_{s \varepsilon S} P(o' \mid a, b)$$

In words, the belief-state in proposed POMDP model can be updated as probability of current observation probabilities of individual services divided by probability of previous observation probabilities of individual services. For example, In Figure 3.3 b_{s3}, b_{s4} and b_{s5} can be computed as follows:

$$b_{s3} = {}^{1}/_{n}\left(\frac{p2}{MAX(p1,p2)}\right), b_{s4} = {}^{1}/_{n}\left(\frac{q2}{MAX(q1,q2)}\right), b_{s5} = {}^{1}/_{n}\left(\frac{r2}{MAX(r1,r2)}\right)$$

where n is the number of services in the list with observation probability not equal to 1. Policies are defined such that the service with higher belief-state value is preferred over others. The observation probability of services $S3$ and $S4$ changes from $p1\ to\ p2$ and $q1\ to\ q2$ respectively. Also action $a2$ can also gather information about service $S5$ with observation probability as $r2$.

3.4 Design of Algorithms

3.4.1 Overview

The *AutomateREinSPL* algorithm is the main algorithm and it is composed of smaller algorithms – *StateEstimator, TrendAnalysis, ModeSelector, PolicySelector* and *MakeAction*

which are explained in greater details in next 5 sections. The Flow for the *AutomateREinSPL* is illustrated in Figure 3.2. The *AutomateREinSPL* algorithm is as follows:

Algorithm 1: AutomateREinSPL

INPUT:
OUTPUT:

```
1.  isGoalState ← false                          // initializing goal to false
2.  belief ← 1                                   // initializing belief value to 1
3.  CREATE empty LIST b_hist
4.  ADD belief to LIST b_hist                    // adding initial belief value to list
5.  WHILE isGoalState NOT EQUAL true             // repeats until goal is reached
6.      input ← READ(observation)
7.      IF input EQUAL 'exit' THEN               // checking for the goal reached?
8.          isGoalState ← true
9.      ELSE
10.         b(s') ← StateEstimator(input, belief)
11.         ADD b(s') to LIST b_hist             // adding update belief value to list
12.         N_cp ← TrendAnalysis(b_hist)
13.         k ← KnowledgeLevelSelector(N_cp)
14.         π_m ← PolicySelector(k)
15.         action ← MakeAction(π_m, b(s'))
16.         belief ← b(s')                       // updating the belief value
17.         PRINT action
18.     ENDIF
19. END WHILE
```

The process starts with initial belief-state value 1 as in Line 2 and creates an empty list to store the belief-state history. The loop in Line 5 is executed until the goal is reached and the status of the goal reached or not is validated in Line 7. A new belief-state value is calculated for the observation received in Line 6 from the *StateEstimator* method and updated in Line 16. The new belief-state value is added to the belief-state history list in Line 11. The sharp variation points (N_{cp}). are obtained from *TrendAnalysis* method in Line 12. The control modes are selected based on N_{cp} from *ModeSelector* method in Line 13 and respective policy π_m is selected through

PolicySelector in Line 14. Finally, Agent will make action based on the policy selected and new belief-state value.

3.4.2 *State Estimator* Module

The State Estimator Module gets the observation as text in natural language. The input text is tokenized using NLP and compared with the service descriptions to identify the services relevant to user input. A new state s' is created and updated belief-state value $b(s')$ is computed using current belief, action taken and observation received as $input$. The *StateEstimator* algorithm is as follows:

Algorithm 2: StateEstimator

INPUT: *input* is string, *belief* is decimal, *action* is string
OUTPUT: *b(s')* is decimal

```
1. tokens ← NLP(input)                    // tokenization using NLP
                                          // A new state s' is create by
2. s' ← MATCH_SERVICES(tokens)            // compares tokens from user input with services
3. b(s') ← Pr(s' | belief, action, input) // calculating new belief using Bayes' rule
4. return b(s')
```

The primary focus of this module is to extract the subjects of user interest using NLP in Line 1 and Line 3 computes the new belief-state value for state s' using Bayes theorem as described in section 2.1.3. The probabilities required to compute belief-state value such as observation probabilities and transition probabilities are determined in Line 2 by matching extracted subject areas in user input with service descriptions.

3.4.3 *Trend Analysis* Module

The Trend Analysis Module gets the belief-state history list as input from *AutomateREinSPL* Module. Each value in the list constitutes of belief-state value and time at which the belief-state value is calculated during the interaction. The output of Trend Analysis Module is number of sharp variation points (N_{cp}) i.e. the coefficients of DWT as described in section 2.2.3.2. The *TrendAnalysis* algorithm is as follows:

Algorithm 3: TrendAnalysis

INPUT: b_{hist} is list
OUTPUT: N_{cp} is integer

1. N_{cp} ← DWT(b_{hist}) // Computing the sharp variation points
2. return N_{cp} // using DWT equation

3.4.4 *Knowledge Level Selector* Module

The Knowledge Level Selector Module gets the sharp variation points (N_{cp}) as input and return k as output. The *KnowledgeLevelSelector* algorithm is as follows:

Algorithm 4: KnowledgeLevelSelector

INPUT: N_{cp} is integer
OUTPUT: k is string

1. expertThreshold ← READ_FROM_TRAINED_MODEL
2. professionalThreshold ← READ_FROM_TRAINED_MODEL
3. amateurThreshold ← READ_FROM_TRAINED_MODEL
4. noviceThreshold ← READ_FROM_TRAINED_MODEL
5. IF N_{cp} < expertThreshold THEN
6. k ← 'expert'
7. ELSE IF N_{cp} >= expertThreshold AND N_{cp} < professionalThreshold
8. k ← 'professional'
9. ELSE IF N_{cp} >= professionalThreshold AND N_{cp} < amateurThreshold
10. k ← 'amateur'
11. ELSE
12. k ← 'novice'
13. ENDIF
14.
15. return k

Output values of k are as follows: expert, professional, amateur and novice. The output mode is selected based on the value of N_{cp} from the conditional statements from Line 5 to Line 11. The thresholds in the algorithm: expertThreshold, professionalThreshold, amateurThreshold and noviceThreshold are constants in this module and are calculated by training the proposed model with different knowledge base users datasets. The methodology involved in training is explained in detail in section 3.6.

3.4.5 *Policy Selector* Module

The Policy Selector module get the $mode$ as input and return respective policy as output. Policies are mapped from belief-state values to action as described in section 2.1.3. Policies are defined using Policy Graph concept and is explained in Chapter 4. The *PolicySelector* Algorithm is as follows:

Algorithm 5: PolicySelector

INPUT: k is string
OUTPUT: π is string

```
1. CASE k OF
2.     expert: return π_expert
3.     professional: return π_professional
4.     amateur: return π_amateur
5.     novice: return π_novice
6. ENDCASE
```

The CASE statement block from Line 1 to Line 6 determines the policy to be returned based on the input k.

3.4.6 *Make Action* Module

The Make action module receives the new belief-state value and policy and returns the $action$ as string to the user. The $actions$ returned can be goal driven or information gathering responses. The *MakeAction* Algorithm is as follows:

Algorithm 6: MakeAction

INPUT: π_m is string, b(s') is decimal
OUTPUT: *action* is string

1. action ← GET(Action from Transition of state *s* to *s'*)
2. return action

3.5 Time Complexity

The time complexities of different modules are as mentioned in the below table.

Modules	Time COMPLEXITY	Notes
State Estimator	Tokenizing - $O(n^2)$ Services Matching - $O(n * m)$	'n' is input string length 'n' is input string length 'm' is number of services
Trend Analysis	Wavelet Transform - $O(n^2)$ Sharp Variation Points Detection - $O(n)$	'n' is belief history length
Knowledge Level Selector	$O(1)$	
Policy Selector	$O(1)$	
Make Action	$O(1)$	

Table 3.1: Time Complexity of *AutomateREinSPL* algorithm

The *StateEstimator* Module uses the libraries defined in section 4.2 so the time complexity involved is $O(n^2)$ for tokenizing [16] and $O(n * m)$ for matching services [16]. The

TrendAnalysis module take the time complexity of $O(n^2)$ for Wavelet Transform [15] and $O(n)$ for detecting sharp variation points. *KnowledgeLevelSelector*, *PolicySelector* and *MakeAction* modules have CASE statements or IF-ELSE ladder and the time complexity is $O(1)$.

3.6 Training POMDP Model for Knowledge Level Thresholds

Training Datasets (5 samples) are prepared manually as described in [9] [19] such that the datasets behave as virtual users with knowledge as expert, professional, amateur and novice.

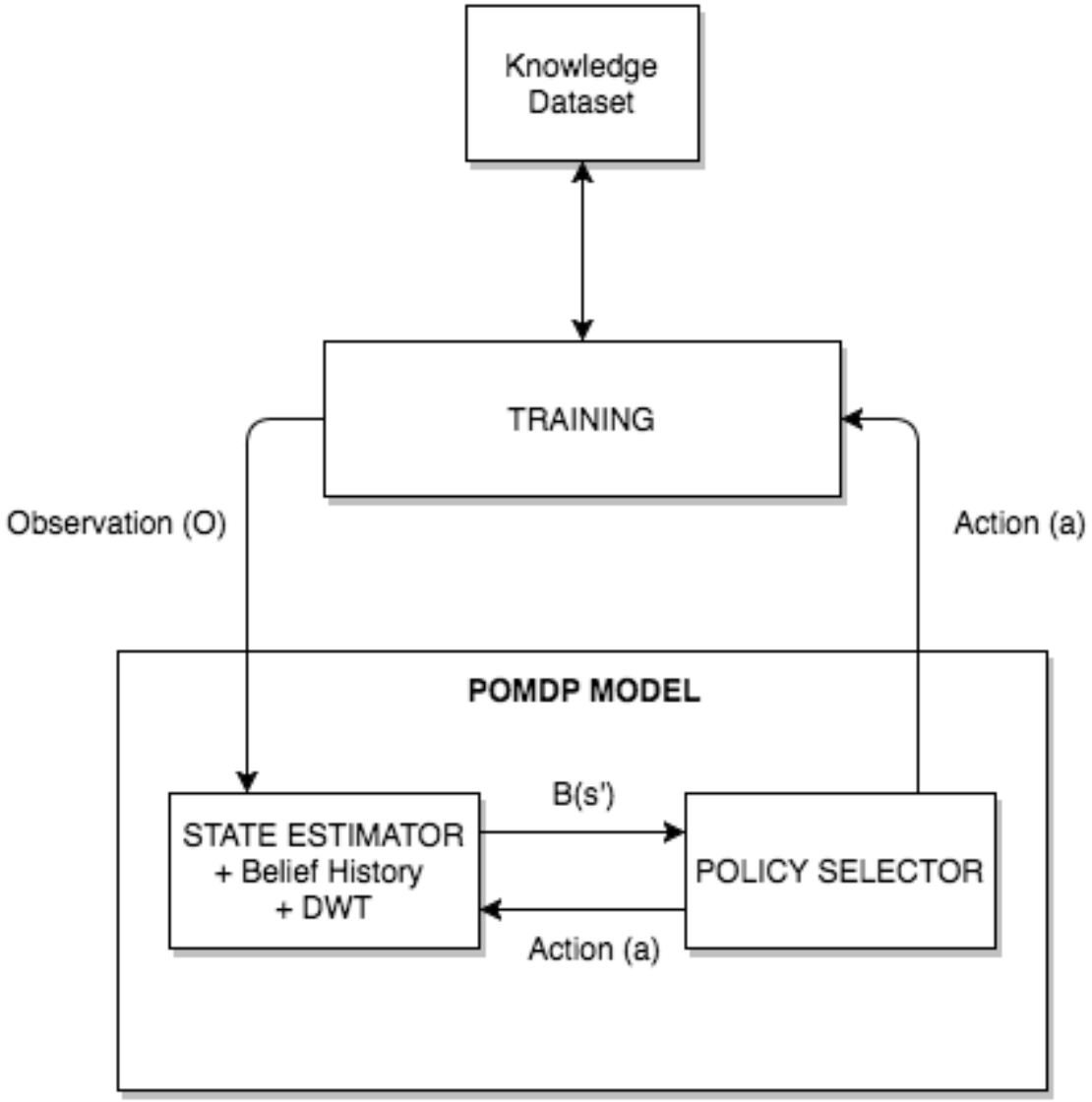


Figure 3.4: Knowledge Trainer Architecture

As illustrated in the Figure 3.4, knowledge dataset is replaced with expert dataset and Policy selectors are replaced with expert policies to determine the range of sharp-variation points. Similarly, the sharp-variation points range is determined for professional, amateurs and novice by replace the respective dataset and policies. The knowledge trainer is executed 1000 times to achieve multiple goals. Goals are the list of services the trainer is intended to achieve. The computed values are used in *KnowledgeSelector* to switch between different policy sets. The minimum and maximum sharp variation points obtained for different knowledge levels are as follows:

Knowledge Level	**Minimum Sharp Variation Points**	**Maximum Sharp Variation Points**
Expert	2	11
Professional	7	15
Amateur	13	21
Novice	17	39

Table 3.2: Min/Max Sharp variation points for different knowledge level

3.7 A Walk Through Example

Let's consider the following situation: Current belief-state value = 1. If the welcome action is performed and observation received is as follows, then the observation probabilities and belief-state values are computed as follows:

Action> *Hi, how can I help you today?*

Observation> *user can manage shopping cart?*

Applying NLP on the observation generated the

Service 17: Manage Shopping Cart → 0.22

Service 20: List the items in Cart → 0.07

Belief-state can be computed as follows:

$$b = {}^{1}/_{2}\left(\frac{0.22}{MAX(0.22,1)} + \frac{0.07}{MAX(0.07,1)}\right) = \frac{0.29}{2} = 0.145$$

There the observation probabilities are Ω(S17) = 0.22 and Ω(S20) = 0.07. The new belief-state value is $b = 0.145$

CHAPTER 4

EXPERIMENT DESIGN

4.1 Overview

This chapter discusses the software's used for the experimental design of proposed method. It also covers the necessary training conducted and simulation environments designed to obtain results.

4.2 Software

The following is the comprehensive list of all the libraries and software's used for experimenting the proposed method.

a. Java 7.0 – The programming language
b. WordNet 3.0 – A Lexical database for English language, which is widely used to develop NLP applications.
c. Java WordNet Library (JWNL 1.4)– A Java API for accessing the WordNet relational dictionary.
d. Three.js – A JavaScript 3D library to render 3D objects on web browsers.
e. NetBeans 8.0 – IDE for developing Java Web Applications.
f. Tomcat Server 7.0 – A web server
g. Bootstrap 3.3.7 – A front-end JavaScript and CSS framework.
h. JWave – Open Source Java implementation of Discrete Wavelet Transform (DWT).

4.3 Ontology-based requirement model

A frame-based dialog system is developed by Xieshen Zhang in this thesis [35] which uses the ontology model as the knowledge base. The knowledge base is used to elicit user's demands. An online book shopping system is used by Zhang [35] in his experiments. The approach used to customize software is discussed in Chapter 2 (refer 2.4.1).

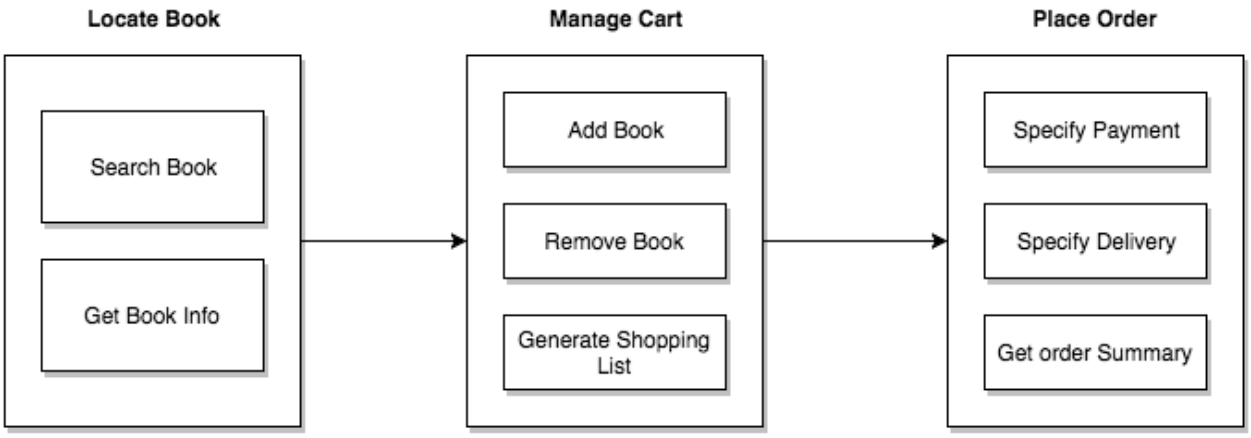
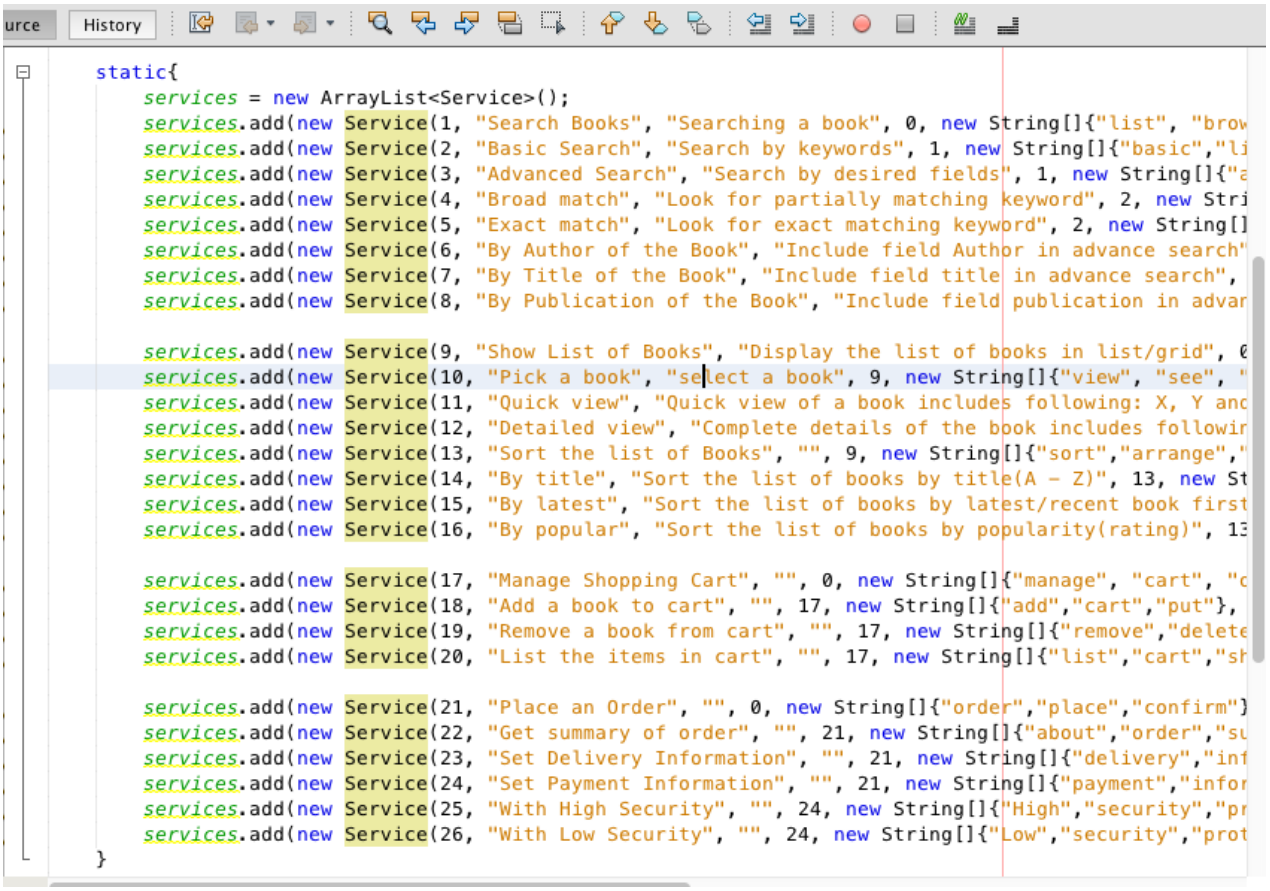


Figure 4.1: Functionalities of an online book shopping system [35]

In this thesis for the experiment design, the subset of ontology is taken from Zhang [35] case study as .OWL file. Figure 4.4 high-level overview of the different services involved. A data model is constructed which stores the services list and dependencies by reading the .OWL file. The data structure for the data model is as follows:

$$Service < serviceId, name, description, parentid, isOptions, hasDependents >$$



```
urce  History
    static{
        services = new ArrayList<Service>();
        services.add(new Service(1, "Search Books", "Searching a book", 0, new String[]{"list", "brow
        services.add(new Service(2, "Basic Search", "Search by keywords", 1, new String[]{"basic","li
        services.add(new Service(3, "Advanced Search", "Search by desired fields", 1, new String[]{"a
        services.add(new Service(4, "Broad match", "Look for partially matching keyword", 2, new Stri
        services.add(new Service(5, "Exact match", "Look for exact matching keyword", 2, new String[]
        services.add(new Service(6, "By Author of the Book", "Include field Author in advance search"
        services.add(new Service(7, "By Title of the Book", "Include field title in advance search",
        services.add(new Service(8, "By Publication of the Book", "Include field publication in advar

        services.add(new Service(9, "Show List of Books", "Display the list of books in list/grid", 0
        services.add(new Service(10, "Pick a book", "select a book", 9, new String[]{"view", "see", "
        services.add(new Service(11, "Quick view", "Quick view of a book includes following: X, Y and
        services.add(new Service(12, "Detailed view", "Complete details of the book includes followir
        services.add(new Service(13, "Sort the list of Books", "", 9, new String[]{"sort","arrange","
        services.add(new Service(14, "By title", "Sort the list of books by title(A - Z)", 13, new St
        services.add(new Service(15, "By latest", "Sort the list of books by latest/recent book first
        services.add(new Service(16, "By popular", "Sort the list of books by popularity(rating)", 13

        services.add(new Service(17, "Manage Shopping Cart", "", 0, new String[]{"manage", "cart", "c
        services.add(new Service(18, "Add a book to cart", "", 17, new String[]{"add","cart","put"},
        services.add(new Service(19, "Remove a book from cart", "", 17, new String[]{"remove","delete
        services.add(new Service(20, "List the items in cart", "", 17, new String[]{"list","cart","sh

        services.add(new Service(21, "Place an Order", "", 0, new String[]{"order","place","confirm"}
        services.add(new Service(22, "Get summary of order", "", 21, new String[]{"about","order","su
        services.add(new Service(23, "Set Delivery Information", "", 21, new String[]{"delivery","inf
        services.add(new Service(24, "Set Payment Information", "", 21, new String[]{"payment","infor
        services.add(new Service(25, "With High Security", "", 24, new String[]{"High","security","pr
        services.add(new Service(26, "With Low Security", "", 24, new String[]{"Low","security","prot
    }
```

Figure 4.2: Data Model for the services

4.4 Interface

A web based application is developed as GUI for user to interact with the ECA. NetBeans 8.0 is used as IDE for developing interface and Apache Tomcat Server is used for hosting the web application. The interface constitutes of four components:

a. *List of Services* – All the available services/assets for automation of SPL are listed. The List of Services are from the data mode discussed in section 4.4.

b. *Selected Services* – All the services/assets selected by user will be listed.

c. *Information Window* – ECA (Agent) communicates with the user through the information window. The information window is located right next to 3D face.

d. *Message Window* – User communicates with the ECA (Agent) by typing message in this window and clicking send button.

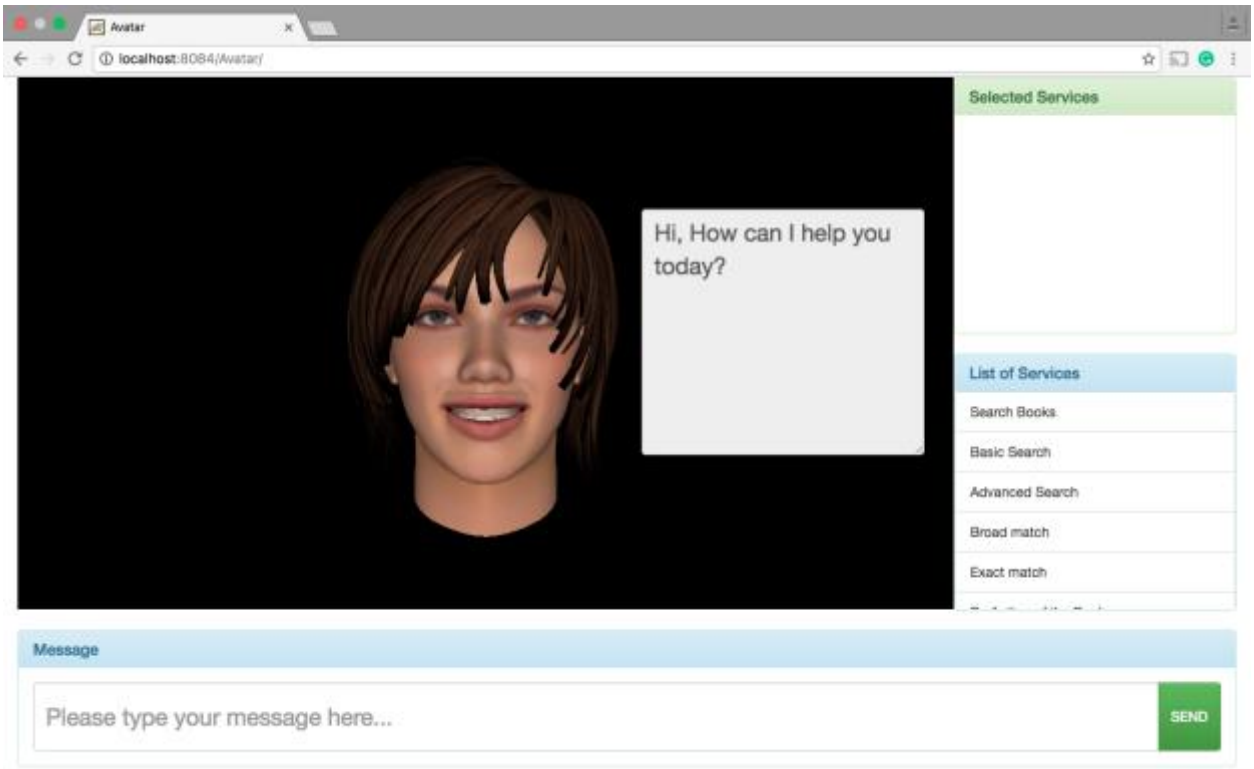


Figure 4.3: Web Application GUI

When user click on the send button an AJAX call (web request to server) happens to post the text in the Message Window to server to process. All the algorithms discussed in the Section 3.3 of Chapter 3 were implemented as illustrated in the Figure 4.7.

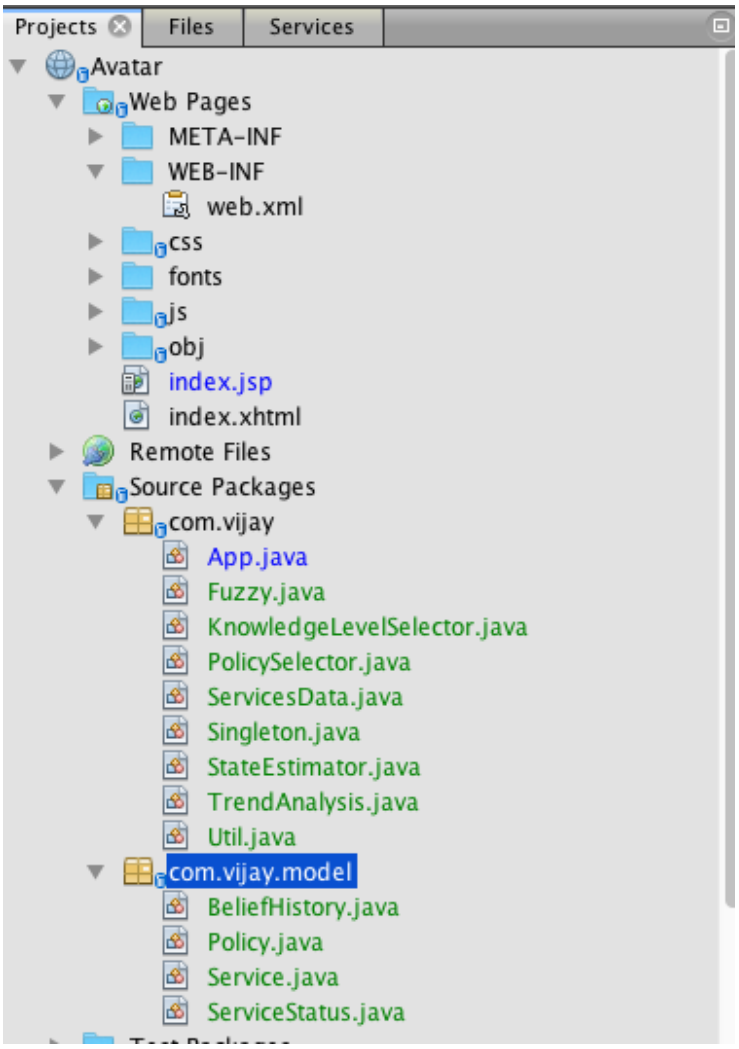


Figure 4.4: Experiment Project Structure

4.5 Simulation Environments

By comparing the results of the proposed POMDP model with the traditional POMDP model, the accuracy of proposed method is evaluated. Therefore, two environments are created to simulate the methods in order to compare the results. Environments are as follows:

a. Using POMDP and Policies – It is a subset of the implementation for the proposed method.

b. Using POMDP, Belief History, DWT and COCOM.

The idea behind creating two environments is to feed both the environments with the same input observations generated from Goal state dataset and semantics of services dataset, to measure the dialog length and accuracy in user intention discovery. The Goal state dataset and semantics of services dataset were explained thoroughly in Chapter 5.

4.5.1 Using POMDP and Policies

As illustrated in the Figure 4.8, the traditional POMDP model constitutes of State Estimator and Policy Selector Modules. The State Estimator module receives the observation and computes the belief-state value. The Policy Selector receives the belief-state value and performs an action.

4.5.2 Using POMDP, Belief-State History, DWT and COCOM

As illustrate in the Figure 4.9, the proposed POMDP model constitutes of all the models explained in the architecture section 3.2.

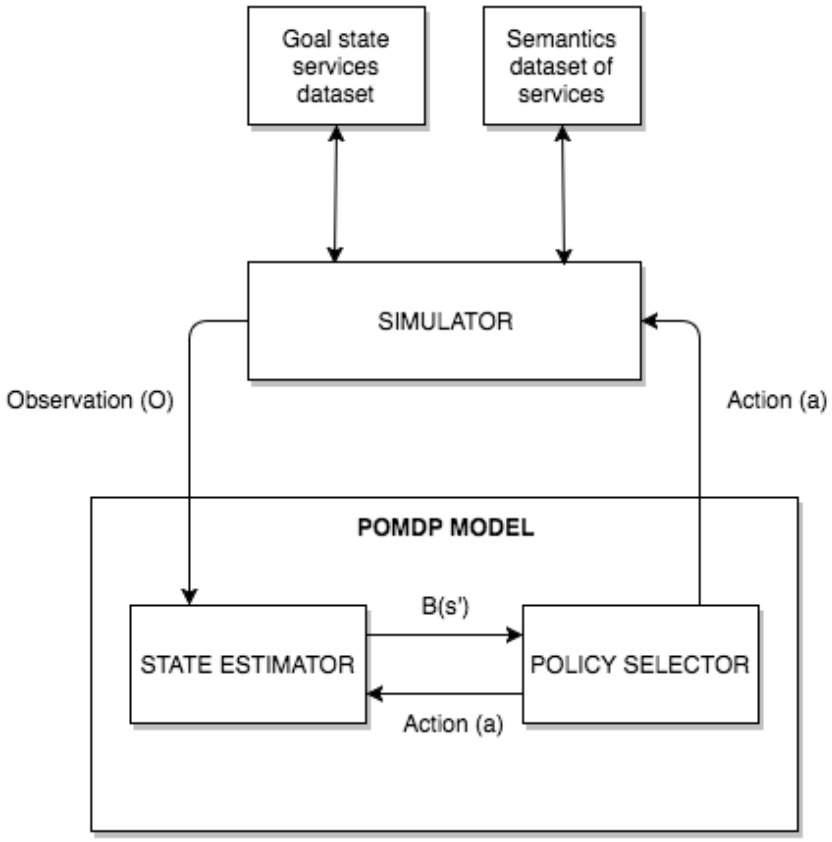


Figure 4.5: Simulator Architecture by using POMDP and Policies

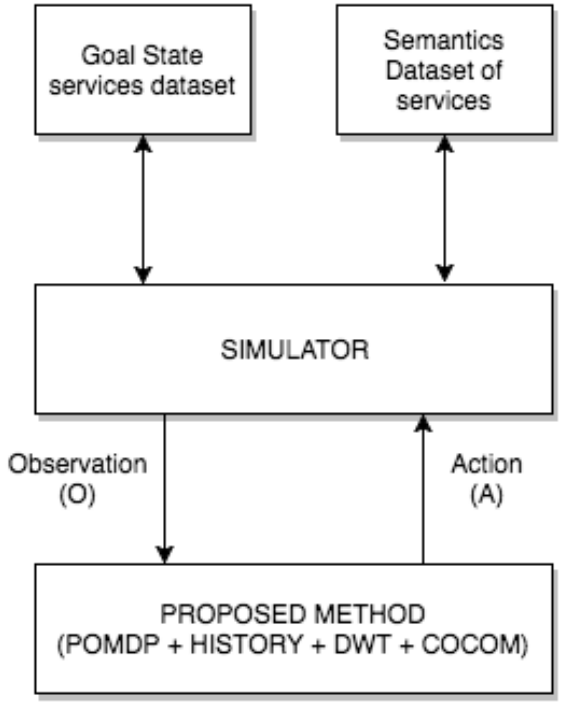


Figure 4.6: Simulator Architecture by using POMDP, History, DWT and COCOM (proposed)

CHAPTER 5

SIMULATION AND RESULTS

5.1 Overview

This chapter discusses how the simulator works and results obtained from the simulation. Before simulator, we will first look at the design of goal and knowledge level semantic datasets.

5.2 Goal Datasets

The simulator imitates the behavior of the user. So, the simulator needs to have a fixed goal as the real world user have a goal. By selecting a different combination of services 25 goals are developed. Simulator holds the list of goals and each goal has the list of service ids selected for that goal. Refer section 4.3 for the data structure of Service. Data Structures of Simulator and Goal Datasets is as follows:

```
Simulator {
    List<Goal> goalDataset;
}

Goal {
    List<Integer> serviceIds;
}
```

The goal datasets developed are illustrated in Figure 5.1.

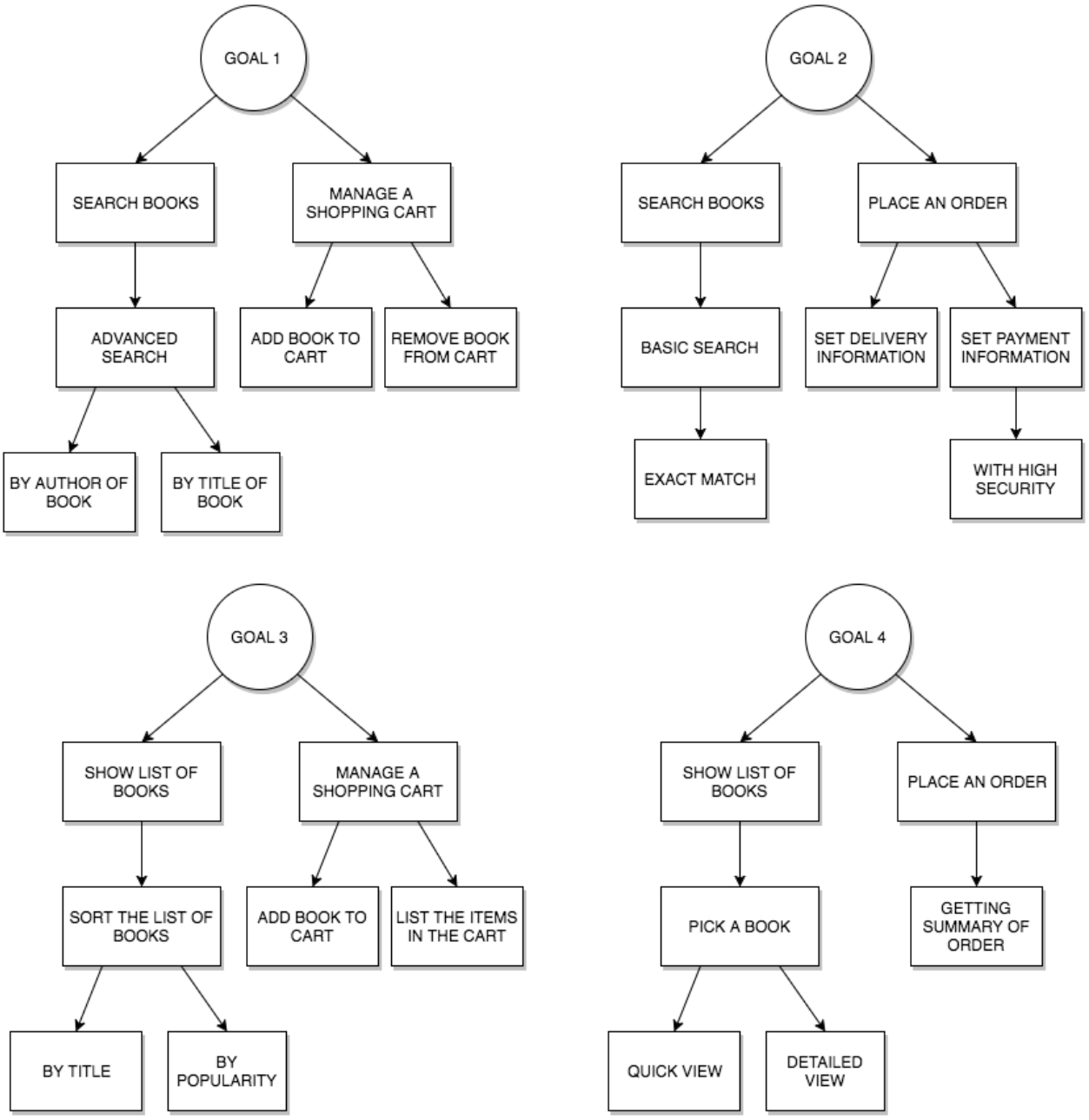


Figure 5.1 Goals created from list of services in Appendix A for simulation

5.3 Knowledge-level semantic datasets

The simulator imitates the behavior of the different user with varying knowledge level. Simulator tests the proposed method by providing observations, affirmative/negative statements

to confirm the actions made by agent. Users are classified as follows: expert, professional, amateur and novice. The strategy for designing the knowledge level semantic datasets are as follows [30]:

a. Expert – Profound knowledge about the requirements both technical and business knowledge.
b. Professional – Have both technical and business knowledge but missing logical implication within the requirements.
c. Amateur – Only have business knowledge and no technical knowledge about the requirements.
d. Novice – No technical and business knowledge about the requirements.

The data structure for simulating users with different knowledge level are as follows:

Expert {

List<String> observations;

}

Novice {

List<String> observations;

}

For Example, consider *Service 20: List the items in the cart*. For the service, *List* is the technical term and *Cart* is the business term. The dataset is as follows:

Expert> List the items in the cart

Professional> Display the books added in the cart

Amateur> User should be able to browse the books in cart

Novice> Display the books from which I can place an order

5.4 Life Cycle of Simulator

The steps involved in simulator are as follow:

Step 1: Select Goal and Knowledge level of user from respective datasets

Step 2: Execute Simulation

Step 3: Pass the observation to the Model

Step 4: Model passes the Action to execute.

Step 5: If goal reached, terminate; else repeat Step 2.

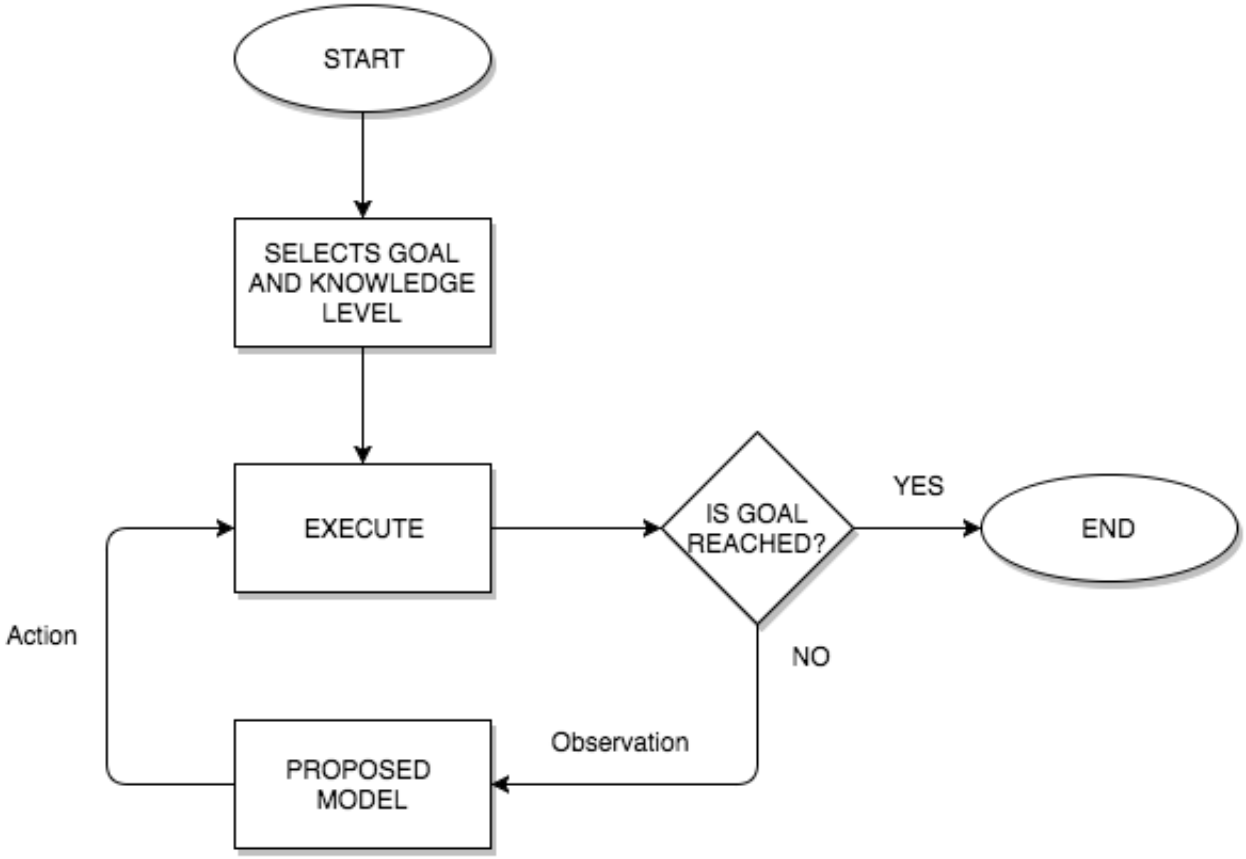


Figure 5.2: Life Cycle of a Simulator

5.5 Results of Simulation

The Traditional POMDP and Proposed POMDP model are tested by randomly selecting a Goal from the Goal datasets and by selecting user knowledge level from knowledge level sematic dataset as explained above. This process is continued for 1000 runs in order to see how many times goal is achieved by the user. Below are the results of the simulation in perspective of number of times goal achieved:

Case	**Traditional POMDP**	**Proposed POMDP Model**				
		Expert	**Professional**	**Amateur**	**Novice**	**Total**
Average number of times user has achieved the goal per 1000 runs	679	250	211	197	135	793
Accuracy in %	67.9%	100%	84.4%	78.8%	54%	79.3%

Table 5.1: Accuracy results comparison of Traditional POMDP and Proposed POMDP

From the above accuracy results, it is evident that the proposed POMDP Model is more accurate in driving the user to achieve their goals than Traditional POMDP based approach. It is observed that the proposed model works 100% accurate for the Expert users and 54% accurate for Novice users. The increase in the accuracy of proposed POMDP model is due to better understanding of requirements, in other words better user intention discovery. Refer sections 3.7 and 4.3 for examples.

The Traditional POMDP and Proposed POMDP model are tested by randomly selecting a Goal from the Goal datasets and by selecting user knowledge level from knowledge level sematic dataset as explained above. This process is continued for 1000 runs in order to see the dialog length for each run. Below are the results of the simulation illustrating the average dialog length for each run:

Case	**Traditional POMDP**	**Proposed POMDP Model**
Average Dialog length for per 1000 runs	34	21
Average number of times user knowledge selection matched with knowledge level in Model per 1000 runs	-	772 (Accuracy is 77.2%)

Table 5.2: Dialog length results comparison of Traditional POMDP and Proposed POMDP

From the above Simulation results, it is evident that the dialog length for the proposed method is less when compared to Traditional POMDP. The decrease in the dialog length is the combination effect of trend analysis on belief history using DWT and making appropriate actions through four COCOM modes.

In summary through simulation, it is observed that the accuracy of user intention discovery is improved, dialog length is decreased and proposed method is successful in addressing users with different knowledge levels.

CHAPTER 6

CONCLUSIONS

6.1 Concluding Remarks

In this thesis, the automation of Requirements Elicitation (RE) in Software Product Line (SPL) is reviewed through Online Book Store system. This research has been accomplished in a number of steps. Initially, all the decision-making algorithms that are necessary for the automation and trend analysis approaches were studied thoroughly to understand their limitations. Therefore, a new POMDP model is proposed where trend analysis is performed on belief-state history to anticipate user knowledge level. Upon recognizing the user knowledge level, they are addressed with different set of policies such that appropriate actions were performed through contextual control modes.

Furthermore, goal based and semantic level simulation is performed to evaluate the proposed method. From the results of the simulation it is evident that, user intention discovery is improved and the dialog length is decreased. Also it is evident that different knowledge level of users were addressed differently.

6.2 Future Improvements and Directions

The proposed method is capable of handling the user by matching the requirements with the service descriptions to effectively automate the RE in SPL thereby producing customized software.

- The actions performed by the agent are appropriate but not optimal. In order perform optimal actions, the agent should make actions that maximize the future rewards. The focus of this thesis is to make appropriate/greedy actions.
- By using value iteration functions on updating the policies through learning. We can transform the planning under uncertainty to reinforcement learning which yields overall better results.
- The more the recognition of services from the observations, the better the user intention discovery. Therefore, there is a lot of scope to still improve the accuracy of user intention discovery by using advanced NLP and context-aware algorithms.

REFERENCES

[1] Adarsh, S. and Janga Reddy, M., 2015. Trend analysis of rainfall in four meteorological subdivisions of southern India using nonparametric methods and discrete wavelet transforms. International Journal of Climatology, 35(6),1107-1124.

[2] Bui, T.H., Zwiers, J., Poel, M. and Nijholt, A., 2006. Toward affective dialogue modeling using partially observable Markov decision processes.

[3] Bui, T.H., Zwiers, J., Poel, M. and Nijholt, A., 2010. Affective dialogue management using factored POMDPs. In Interactive Collaborative Information Systems (pp. 207-236). Springer Berlin Heidelberg.

[4] Brillinger, D.R., 2001. Time series: data analysis and theory, Siam,46.

[5] Cassell, J., 2000. Embodied conversational interface agents. Communications of the ACM, 43(4), 70-78.

[6] Cassell, J., Bickmore, T., Vilhjálmsson, H. and Yan, H., 2000, January. More than just a pretty face: affordances of embodiment. In Proceedings of the 5th international conference on Intelligent user interfaces (pp. 52-59). ACM.

[7] Cormode, G., Garofalakis, M., Haas, P.J. and Jermaine, C., 2012. Synopses for massive data: Samples, histograms, wavelets, sketches. Foundations and Trends in Databases, 4(1–3),1-294.

[8] Creed, C., Beale, R. and Cowan, B., 2015. The Impact of an Embodied Agent's Emotional Expressions Over Multiple Interactions. Interacting with Computers, 27(2),172-188.

[9] Dhanapal, R., 2012. An Approach for Contextual Control in Dialogue Management with Belief State Trend Analysis and Prediction.

[10] Flycht-Eriksson, A. and Jönsson, A., 2000, October. Dialogue and domain knowledge management in dialogue systems. In Proceedings of the 1st SIGdial workshop on Discourse and dialogue. Association for Computational Linguistics.10,121-130.

[11] Gomaa, H. and Shin, M.E., 2007, January. Automated software product line engineering and product derivation. In System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on IEEE,258a.

[12] Goguen, J.A. and Linde, C., 1993. Techniques for requirements elicitation.RE, 93,152-164.

[13] http://www.sei.cmu.edu/productlines/ [Last Accessed on 22/07/2016]

[14] http://pomdp.org [Last accessed on 15/06/2016]

[15] https://code.google.com/p/jwave/ [Last accessed on: 20/07/2016]

[16] https://sourceforge.net/projects/jwordnet/ [Last accessed on 07/08/2016]

[17] `Krueger, C., 2001, October. Easing the transition to software mass customization. In International Workshop on Software Product-Family Engineering. Springer Berlin Heidelberg,282-293.

[18] Keskin, C., Balci, K., Aran, O., Sankur, B. and Akarun, L., 2007, May. A multimodal 3D healthcare communication system. In 3D Conference.

[19] Kaler, M.S.2014. An interactive approach to software visualization for customization. Electronic Master's Theses and Dissertations, CS Dept. UWindsor, Windsor, ON, 5161, 1-102.

[20] Kotonya, G. and Sommerville, I., 1998. Requirements engineering: processes and techniques. Wiley Publishing.

[21] Mimoun, M.S.B., Poncin, I. and Garnier, M., 2012. Case study—Embodied virtual

agents: An analysis on reasons for failure. Journal of Retailing and Consumer services, 19(6),605-612.

[22] Pohl, K., Böckle, G. and van Der Linden, F.J., 2005. Software product line engineering: foundations, principles and techniques. Springer Science & Business Media.

[23] Polikar, R., 1999. The story of wavelets. Physics and modern topics in mechanical and electrical engineering,192-197.

[24] Polikar, R., 1996. The wavelet tutorial.

[25] Rabiser, R., Grünbacher, P. and Dhungana, D., 2010. Requirements for product derivation support: Results from a systematic literature review and an expert survey. Information and Software Technology, 52(3), 324-346.

[26] Rickenberg, R. and Reeves, B., 2000, April. The effects of animated characters on anxiety, task performance, and evaluations of user interfaces. In Proceedings of the SIGCHI conference on Human Factors in Computing Systems ,ACM,49-56.

[27] Sadri, V.2012. A Petri-Net Based Approach of Software Visualization for Software Customization. Electronic Master's Theses and Dissertations, CS Dept. UWindsor, Windsor, ON.

[28] Sriram, S., 2012. Improved Intention Discovery with Classified Emotions in A Modified POMDP.

[29] Sommerville, I. and Sawyer, P., 1997. Requirements engineering: a good practice guide. John Wiley & Sons, Inc.

[30] Stanton, N.A., Ashleigh, M.J., Roberts, A.D. and Xu, F., 2001. Testing Hollnagel's contextual control model: Assessing team behaviour in a human supervisory control task. Journal of Cognitive Ergonomics, 5(1), 21-33.

[31] Tripathi, S., 2016. A Run-Time Approach of Combining Ontologies to Enhance Interactive Requirements Elicitation for Software Customization.

[32] Tsui, F., Karam, O. and Bernal, B., 2013. Essentials of software engineering. Jones & Bartlett Publishers.

[33] Yuan, X. and Bian, L., 2010, December. A modified approach of POMDP-based dialogue management. In Robotics and Biomimetics (ROBIO), 2010 IEEE International Conference,816-821.

[34] Yuan, X. and Vijayarangan, R., 2013, August. Emotion Animation of Embodied Conversational Agents with Contextual Control Model. In Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing,IEEE,670-677.

[35] Yuan, X. and Zhang, X., 2013, August. An interactive approach of online software customization via conversational Web agents. In Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing ,IEEE,327-334.

[36] Yuan, X. and Bian, L., 2010, December. A modified approach of POMDP-based dialogue management. In Robotics and Biomimetics (ROBIO), 2010 IEEE International Conference on IEEE,816-821.

[37] Yuan, X. and Zhang, X., 2015, August. An ontology-based requirement modeling for interactive software customization. In Model-Driven Requirements Engineering Workshop (MoDRE), 2015 IEEE International (pp. 1-10). IEEE.

[38] Zhang, M., 2016. Real-time Traffic Flow Prediction using Augmented Reality.

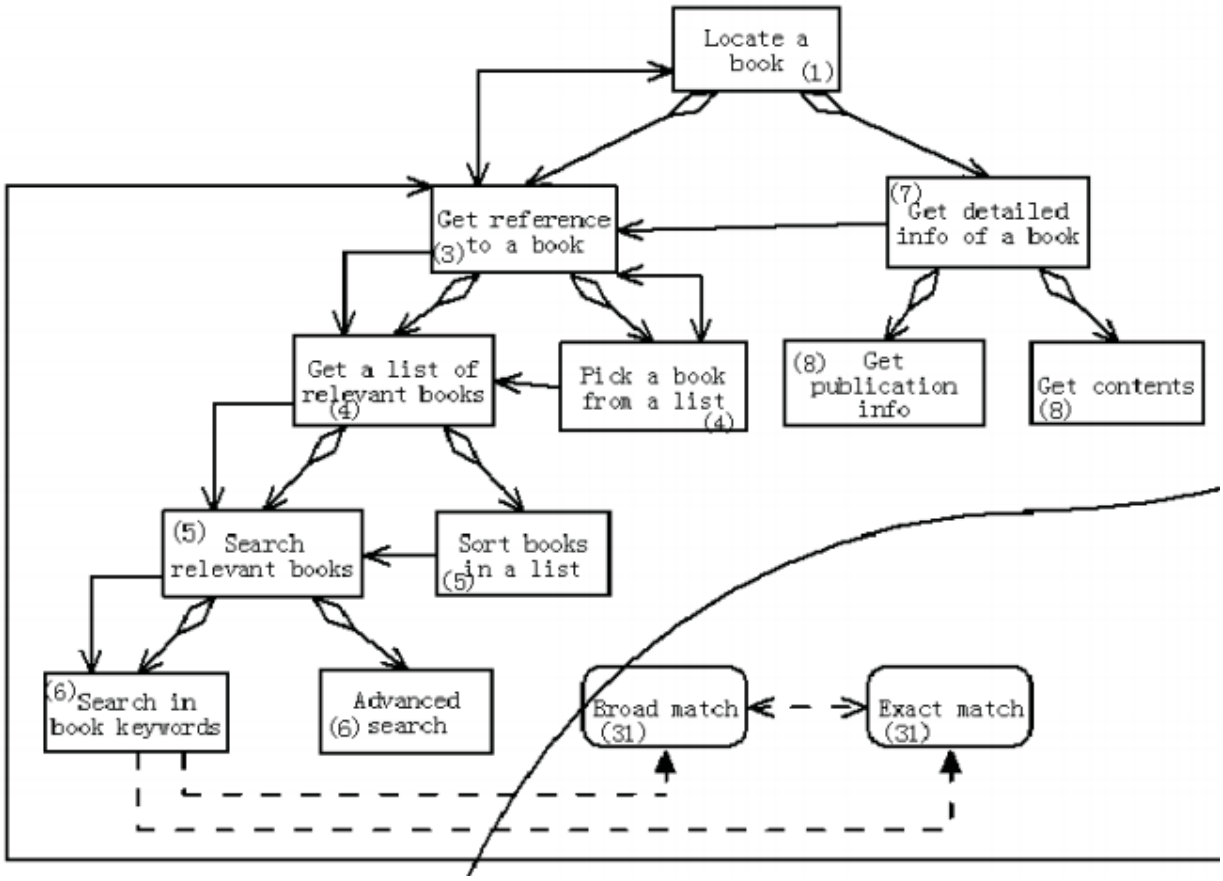
[39] Zhang, X., 2011. An interactive approach of ontology-based requirement elicitation for software customization.

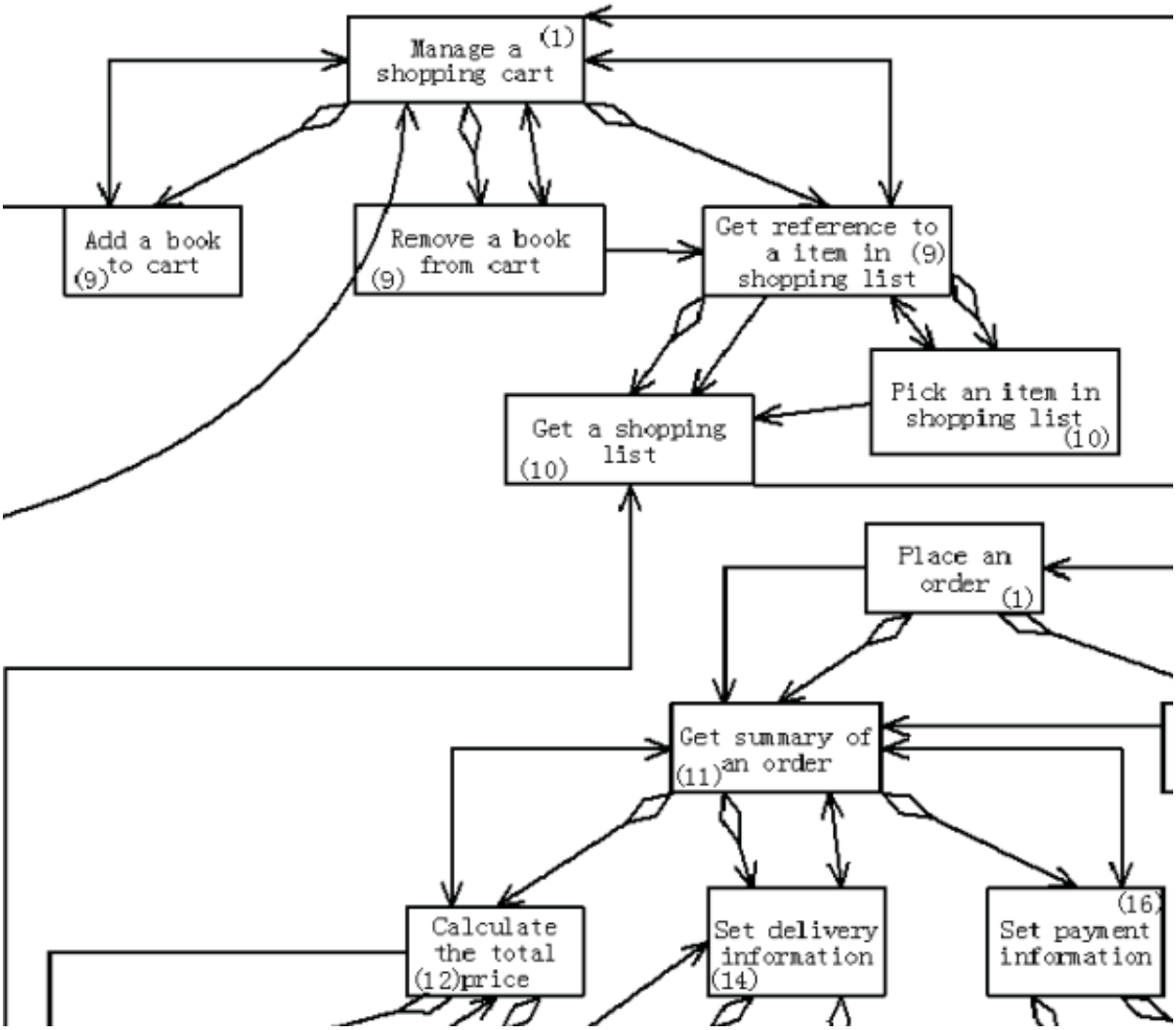
APENDICES

Appendix A

The Subset of Requirement Model from Zhang [35] case study

The following figure illustrates the complete ontology-based requirement model instantiated with the case study of online book shopping service. The figure is divided into four parts





List of Services:

1. Search Books

    1.1. Basic Search

        1.1.1. Broad Match

        1.1.2. Exact Match

    1.2. Advanced Search

        1.2.1. By author of book

1.2.2. By title of book

1.2.3. By publication of book

2. Show list of books

2.1. Pick a book

2.1.1. Quick View

2.1.2. Detailed View

2.2. Sort the list of books

2.2.1. By title

2.2.2. By latest

2.2.3. By popularity

3. Manage a Shopping Cart

3.1. Add a Book to cart

3.2. Remove a Book from cart

3.3. List the items in the cart

4. Place an Order

4.1. Getting summary of order

4.2. Set Delivery Information

4.3. Set Payment Information

4.3.1. With High Security

4.3.2. With Low Security

Appendix B

The Following is the Java code for the AutomateREinSPL Algorithm in section 3.4.1

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package com.vijay;

import com.vijay.model.BeliefHistory;
import com.vijay.model.Policy;
import edu.stanford.nlp.ling.CoreAnnotations.LemmaAnnotation;
import edu.stanford.nlp.ling.CoreAnnotations.SentencesAnnotation;
import edu.stanford.nlp.ling.CoreAnnotations.TextAnnotation;
import edu.stanford.nlp.ling.CoreAnnotations.TokensAnnotation;
import edu.stanford.nlp.ling.CoreLabel;
import edu.stanford.nlp.pipeline.Annotation;
import edu.stanford.nlp.pipeline.StanfordCoreNLP;
import edu.stanford.nlp.simple.*;
import edu.stanford.nlp.util.CoreMap;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.List;
import java.util.Properties;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import net.didion.jwnl.JWNL;

/**
 *
 * @author vijaymulpuri
 */
public class App extends HttpServlet {

    /**
     * Processes requests for both HTTP <code>GET</code> and <code>POST</code>
     * methods.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        String message = request.getParameter("message");
        message = message.trim();
        String result = process(message);
        System.out.println(result);
        try (PrintWriter out = response.getWriter()) {
            /* TODO output your page here. You may use following sample code. */
            out.println(result);

        }
    }

    public String process(String input){

        double newBelief = StateEstimator.process(input);
        BeliefHistory historyInstance = BeliefHistory.getInstance();
```

```
        historyInstance.addHistory(newBelief);
        int ncp = TrendAnalysis.process(historyInstance.getHistory());
        String k = KnowledgeLevelSelector.process(ncp);
        Policy policy = PolicySelector.process(k, newBelief);

        String[] expressions = {"A", "D", "F", "H", "SA", "SU"};
        double[] expressionCoeff = Fuzzy.process(ncp, policy.getReward());
        String expModel = "";
        for(int i=0;i<expressions.length;i++){
            if(expressionCoeff[i]>0.0){
                expModel += expressions[i]+"_";
            }
        }
        expModel = "H_SU";//expModel.substring(0, expModel.length()-1);

        return policy.getAction();//"{'action':'"+policy.getAction()+"', 'expression':'"+expModel+"'}";
    }

    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the + sign on the left to edit the code.">
    /**
     * Handles the HTTP <code>GET</code> method.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Handles the HTTP <code>POST</code> method.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Returns a short description of the servlet.
     *
     * @return a String containing servlet description
     */
    @Override
    public String getServletInfo() {
        return "Short description";
    }// </editor-fold>

}
```

Appendix C

The Following is the HTML, CSS and JS code for the Interface described in section 4.5

```
<%--
    Document   : index
    Created on : Jul 23, 2016, 12:13:41 PM
    Author     : vijaymulpuri
--%>

<%@page import="java.util.ArrayList"%>
<%@page import="com.vijay.ServicesData"%>
<%@page import="com.vijay.model.Service"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <meta name="viewport" content="width=device-width, user-scalable=no, minimum-scale=1.0, maximum-scale=1.0">
        <title>Avatar</title>
        <!-- Latest compiled and minified CSS -->
        <link rel="stylesheet" href="css/bootstrap.min.css">

        <!-- Optional theme -->
        <link rel="stylesheet" href="css/bootstrap-theme.min.css">
        <link rel="stylesheet" type="text/css" href="css/avatar.css"/>
        <%
            ArrayList<Service> servicesList = ServicesData.getServices();
        %>
    </head>
    <body>
        <div class="container-fluid fill" >
            <div class="row" id="mainWindow">
                <div class="col-md-6">
                    <div id="avatar"></div>
                </div>
                <div class="col-md-3">
                    <textarea rows="7" class="form-control" id="info" name="info" style="font-size: 24px;" disabled></textarea>
                </div>
                <div class="col-md-3">
                    <div class="panel panel-success" id="servicesSelectedWindow">
                        <div class="panel-heading">
                            <h3 class="panel-title">Selected Services</h3>
                        </div>
                        <div class="panel-body" style="padding:0px;">
                        </div>
                    </div>
                    <div class="panel panel-info" id="servicesListWindow">
                        <div class="panel-heading">
                            <h3 class="panel-title">List of Services</h3>
                        </div>
                        <div class="panel-body" style="padding:0px;">
                            <ul class="list-group">
                                <%
                                    for(Service service: servicesList){
                                %>
                                <li class="list-group-item"><%=service.getName()%></li>
                                <%
```

```
                        }
                    %>
                  </ul>
                </div>
              </div>
            </div>
          </div>
          <div class="row" id="messageWindow">
            <div class="col-md-12">
              <div class="panel panel-info">
                <div class="panel-heading">
                  <h3 class="panel-title">Message</h3>
                </div>
                <div class="panel-body">
                  <div class="input-group">
                    <input class="form-control" id="message" name="message" style="height: 70px;font-size: 24px;" placeholder="Please type your message here..."/>
                    <span class="input-group-btn">
                      <button class="btn btn-success" style="height: 70px;" id="send">SEND</button>
                    </span>
                  </div>
                </div>
              </div>
            </div>
          </div>
        </div>

        <!-- Latest compiled and minified JavaScript -->
        <script src="https://code.jquery.com/jquery-2.2.4.min.js" integrity="sha256-BbhdlvQf/xTY9gja0Dq3HiwQF8LaCRTXxZKRutelT44=" crossorigin="anonymous"></script>
        <script src="js/bootstrap.min.js"></script>
        <script src="js/three.min.js"></script>
        <script src="js/DDSLoader.js"></script>
        <script src="js/MTLLoader.js"></script>
        <script src="js/OBJLoader.js"></script>
        <script src="js/Detector.js"></script>
        <script src="js/stats.min.js"></script>
        <script>

          var container, stats;

          var camera, scene, renderer;

          var mouseX = 0, mouseY = 0;

          var windowHalfX = window.innerWidth / 2;
          var windowHalfY = window.innerHeight / 2;
          var initialExpression = "H";              // Happy

          loadExpression(initialExpression);
          animate();

          function loadExpression(expression) {

            container = document.getElementById("avatar");

            camera = new THREE.PerspectiveCamera(45, window.innerWidth / window.innerHeight, 1, 2000);
            camera.position.z = 250;

            // scene
```

```
    scene = new THREE.Scene();

    var ambient = new THREE.AmbientLight(0x444444);
    scene.add(ambient);

    var directionalLight = new THREE.DirectionalLight(0xffeedd);
    directionalLight.position.set(0, 0, 1).normalize();
    scene.add(directionalLight);

    // model

    var onProgress = function (xhr) {
      if (xhr.lengthComputable) {
        var percentComplete = xhr.loaded / xhr.total * 100;
        console.log(Math.round(percentComplete, 2) + '% downloaded');
      }
    };

    var onError = function (xhr) { };
    THREE.Loader.Handlers.add(/\.dds$/i, new THREE.DDSLoader());

    var mtlLoader = new THREE.MTLLoader();
    mtlLoader.setPath('obj/');
    mtlLoader.load('neutral.mtl', function (materials) {

      materials.preload();

      var objLoader = new THREE.OBJLoader();
      objLoader.setMaterials(materials);
      objLoader.setPath('obj/');
      objLoader.load(expression+'.obj', function (object) {

        object.position.z = -300;
        scene.add(object);

      }, onProgress, onError);

    });

    //

    renderer = new THREE.WebGLRenderer();
    renderer.setPixelRatio(window.devicePixelRatio);
    renderer.setSize(3 * window.innerWidth / 4, 3 * window.innerHeight / 4);
    while (container.hasChildNodes()) {
      container.removeChild(container.lastChild);
    }
    container.appendChild(renderer.domElement);

    //document.addEventListener('mousemove', onDocumentMouseMove, false);

    //

    window.addEventListener('resize', onWindowResize, false);

}

function onWindowResize() {

    windowHalfX = window.innerWidth / 2;
    windowHalfY = window.innerHeight / 2;
```

```
    camera.aspect = window.innerWidth / window.innerHeight;
    camera.updateProjectionMatrix();

    renderer.setSize(3 * window.innerWidth / 4, 3 * window.innerHeight / 4);

}

function onDocumentMouseMove(event) {

    mouseX = (event.clientX - windowHalfX) / 2;
    mouseY = (event.clientY - windowHalfY) / 2;

}

//

function animate() {

    requestAnimationFrame(animate);
    render();

}

function render() {

    camera.position.x += (mouseX - camera.position.x) * .05;
    camera.position.y += (-mouseY - camera.position.y) * .05;
    camera.lookAt(scene.position);

    renderer.render(scene, camera);

}

$(document).ready(function () {
    adjustHeight();
    document.getElementById("info").value += "Hi, How can I help you today?\n";
    $("button").click(function () {
        execute();
    });
    $("#message").keypress(function (e) {
        if (e.which === 13) {  //checks whether the pressed key is "Enter"
            execute();
        }
    });
});

function execute(){
    $.ajax({
        url: "App",
        type: "get", //send it through get method
        data: {message: document.getElementById("message").value},
        success: function (response) {
            //var obj = JSON.parse(response);
            //alert(obj.expression);
            document.getElementById("info").value = response;//obj.action;
            document.getElementById("message").value = "";
            loadExpression("H");
            animate();
        },
        error: function (xhr) {
```

```
                console.log(xhr);
            }
        });
    }

    function adjustHeight(){
        var messageWindow = document.getElementById("messageWindow");
        messageWindow.style.height = "160px";
        console.log(window.innerHeight, messageWindow.offsetHeight);
        var mainWindow = document.getElementById("mainWindow");
        mainWindow.style.height = (window.innerHeight - messageWindow.offsetHeight)+"px";

        var servicesSelectedWindow = document.getElementById("servicesSelectedWindow");
        servicesSelectedWindow.style.height = (mainWindow.offsetHeight/2 - 20)+"px";
        servicesSelectedWindow.style.overflowY = "scroll";
        var servicesListWindow = document.getElementById("servicesListWindow");
        servicesListWindow.style.height = (mainWindow.offsetHeight/2 - 20)+"px";
        servicesListWindow.style.overflowY = "scroll";
        var infoWindow = document.getElementById("info");
        infoWindow.style.margin = (2*servicesListWindow.offsetHeight/3-40)+"px 0px";
    }

  </script>
 </body>
</html>
```

VITA AUCTORIS

NAME: Vijaya Krishna Mulpuri

PLACE OF BIRTH: Guntur, India

YEAR OF BIRTH: 1988

EDUCATION: Bachelor of Technology
Information Technology
SASTRA University, India

Master of Science
Computer Science
University of Windsor, Canada