

Reasoning about Requirements at Run-time Using AI Planning Techniques



By
Zara Hassan
2011-NUST-MS-PhD IT-052

Supervisor
Dr. Nauman Ahmed Qureshi
Department of Computing

A thesis submitted in partial fulfillment of the requirements for the degree
of Masters of Science in Information Technology (MS IT)

In
School of Electrical Engineering and Computer Science,
National University of Sciences and Technology (NUST),
Islamabad, Pakistan.

(September 2014)

Approval

It is certified that the contents and form of the thesis entitled “Reasoning about Requirements at Run-time Using AI Planning Techniques” submitted by Zara Hassan have been found satisfactory for the requirement of the degree.

Advisor: Dr. Nauman Ahmed Qureshi

Signature: _____

Date: _____

Committee Member 1: Dr. Hamid Mukhtar

Signature: _____

Date: _____

Committee Member 2: Dr. Kashif Sharif

Signature: _____

Date: _____

Committee Member 3: Dr. Awais Shibli

Signature: _____

Date: _____

Abstract

The emerging domain of Self-Adaptive Systems (SAS) has gained significant importance in software engineering community over the recent years. Self-adaptive application by definition should modify itself at run-time as a response to the changes in system environment or changes in system\user requirements [14]. Nowadays mobile software applications are being widely used. Such applications must ensure high customizability and at the same time effective reasoning to meet their objectives so that their end-user goals are met. Explicitly, they should be able to: (i) reason about their own requirements and refine and validate them at run-time by involving end-users [16] (ii) provide solutions for the refined or changed requirements at run-time, for instance by using available services [10]. In short, requirements engineering for adaptive mobile applications requires efficient techniques which is a major challenge. In this context, my thesis will focus on extending the reasoning capabilities in Continuous Adaptive RE framework i.e. [10, 16] using AI techniques. In order to understand the aim of this area of research, initially we focused on requirements problem i.e. requirement specification that entails the satisfaction of end user goals at run-time, which is considered as a planning problem. Therefore, methods that can support run-time reasoning of requirements are desirable. We focused on recently proposed techniques for automated reasoning with requirements goals and preference models to support run-time adaptations. These techniques are integrated into CARE (Continuous Adaptive RE), [10] which is a framework for requirements based engineering of self-adaptive system and continuous reappraisal of adaptive requirements at run-time.

CARE framework has different components. In this thesis work, we are working on reasoning component of CARE framework, which provides effective decision making based on end-user preferences and goal models, supported by AI planning techniques subsequently providing reasoning about new solutions to the requirements problem. We envision that the utmost advantages of our approach are that the decision making process of self-adaptive system align directly with human accessible requirements models, facilitating thereby systematic engineering and accessibility both at design time and at run-time [10].

Certificate of Originality

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Author Name: Zara Hassan

Signature: _____

Acknowledgment

First of all, I am grateful to Allah almighty, who helped me in challenges that were faced while accomplishing this task. He is the One, gave me courage and sustainability whenever I was in need.

I also thank my Family, specially my husband for his support and care at all times. My deepest gratitude goes to my research advisor Dr. Nauman Ahmed Qureshi for his support and guidance. I simply could not think of a more deep-sighted, knowledgeable and practical supervisor, who also understands the psychology of his pupils and introduces to them the finer points of the subject in the easiest possible manner.

I am full of appreciation to the respected committee members Dr. Hamid Mukhtar, Dr. Kashif Sharif and Dr. Awais Shibli for their cooperation and efforts during my thesis.

My SEECS fellows and colleagues duly deserve admiration for backing me up and assisting me in many little ways yet making a huge difference in overall progress.

I dedicate this thesis to my parents, husband and my little son who have been so close to me that I found them with me when I needed. It is their unconditional love that motivates me to set higher targets.

Table of Contents

1	Introduction	1
1.1	Problem Statement . . .	1
1.2	Objective . . .	2
1.3	Relevance to National Needs	2
1.4	Advantages	2
1.5	Areas of Application	3
1.6	Structure	3
2	Literature Survey	4
2.1	Overview . . .	4
2.2	Self-Adaptive systems . . .	4
2.2.1	Modeling dimensions of self-adaptive systems	5
2.3	Self-Management	7
2.3.1	Architectural Model for Self-Managed systems . . .	8
2.4	Requirement engineering of self-adaptive systems	9
2.4.1	Goal Oriented Approach . . .	10
2.4.2	Requirements Monitoring Approaches . . .	11
2.5	Continuous Adaptive Requirements Engineering Framework	11
2.5.1	RE at Design-Time vs RE at Run-Time	12
2.6	Summary	14
3	Proposed Architecture	15
3.1	Proposed Process	15
3.2	CARE Reasoning Methodology	16
3.3	Goal modeling and HTNs	17
3.4	Parsers and AI Planners	18
3.4.1	Planning4J and PDDL4J . . .	18
3.4.2	ANTLR V.3 . . .	19
3.4.3	AI planners	19
3.5	Summary	21

TABLE OF CONTENTS	vii
4 CARE Reasoning Application	22
4.1 Application Scenario	22
4.2 Implementation of CARE Reasoning Framework . . .	23
4.2.1 Techniques of Goal Modeling	23
4.3 Mapping Goal Model to PDDL using HTN Semantics	28
4.3.1 Planning Domain . . .	28
4.3.2 Planning Problem . . .	28
4.4 Development Environment of CARE Reasoning Framework . .	31
5 Evaluation	38
5.1 Evaluation of Reasoning Framework . . .	38
5.1.1 Task vs Time	39
5.1.2 Evaluation Scenarios	40
6 Conclusion & Future Work	44
6.1 Conclusion . . .	44
6.2 Future work . . .	45

List of Figures

3.1	Modified CARE Reasoning Framework	16
4.1	Modelling Tools . . .	24
4.2	Working Day Module . . .	25
4.3	Holiday Module . . .	26
4.4	Pesudo Code for Planning Domain (Transportation Scenario) .	29
4.5	Pesudo Code for Planning Domain (Packing Bag Scenario) .	29
4.6	Pesudo Code for Planning Domain (Daily Job Tasks)	30
4.7	Pesudocode for User Planning Problem 1 . . .	30
4.8	Pesudocode for User Planning Problem 2 . . .	31
4.9	Pesudocode for User Planning Problem 3 . . .	31
4.10	Algorithm for Getting Ready Module . . .	32
4.11	Algorithm for Transportation Module . . .	32
4.12	Algorithm for Holiday Module . . .	33
4.13	Algorithm for Holiday Actions Module	34
4.14	User Profile	35
4.15	Transportation Module 1 . . .	35
4.16	Transportation Module 2 . . .	36
4.17	Holiday Actions Module	36
4.18	Holiday Actions Module	37
5.1	Tasks vs Time Trend . . .	39
5.2	Plan1	40
5.3	Plan2	41
5.4	Plan3	42
5.5	Comparison of scenarios	43

Chapter 1

Introduction

Change is inevitable for any developed software. End-user preferences vary with respect to time. Moreover, software that is built for end-users encounters dynamic change requests for which software needs to cater for such change during its course of execution. With this premise, we focus on requirements which are expressed as goals. This gives rationale to change or reconfigure the software to deal with such dynamic changes at run-time. So there is a dire need to introduce systems that respond according to the changed requirements of user and continue working optimally. So capturing and satisfying the user requirements at run-time has posed itself as a new challenge for the software industry. Recent proposals in the area of software engineering of (SAS) self- adaptive systems have argued on methods and techniques needed to ensure such change management. We take the perspective of requirements engineering, since to the best of our knowledge less work has been done on providing reasoning capabilities to the software based on requirements. In this context, a novel framework CARE has been proposed in [2] which is a framework for requirements based engineering of self-adaptive systems and continuous reappraisal of adaptive requirements at run-time. The architecture for CARE has been proposed already but a lot of work is yet to be undertaken at its implementation level. So in our work we are extending/implementing reasoning component of CARE framework.

1.1 Problem Statement

Extending a reasoning component of CARE framework, which provides effective decision making based on end-user preferences and goal models, supported by AI planning techniques subsequently providing reasoning about new solutions to the requirements problem.

1.2 Objective

1. To focus on reasoning component of CARE framework for automated reasoning of requirements at run-time.
2. To design a system/architecture empowered enough to interactively take input from user and sense the operating environment; and consequently able to assess and reason about both at runtime to generate an intelligent and efficient solution/line of action.
3. To use CARE framework for large problems to estimate its scalability in terms of model complexity, efficiency and meaningfulness of the *reasoning* process.

1.3 Relevance to National Needs

Most of the software failures in our software industry are due to the lack of understanding about requirements engineering for the intended software to be. A lot of work has been done in various fields of software engineering but nowadays requirements engineering has opened new venues for progress. To improve our work regarding different software applications we need to focus on system level requirements and the systems which are automated and self-adaptive.

Software systems such as hospital management system, air traffic control system, service based application etc., requires precision, differently than end-user applications where needs change over time. Our focus is to provide effective methods and techniques to build software which are not only in accordance with respect to requirements but also reason the requirements at run-time such that they are able to meet the goals by adapting themselves.

1.4 Advantages

1. Allows self-adaptation of software applications.
2. Allows reasoning of requirements at Run-time using AI techniques.
3. Allows software applications to behave according to user needs.
4. Enhances productivity by smartly removing the bottle necks by customized software behavior for each user.

1.5 Areas of Application

This research can be applied to almost all areas of software applications in which user goals can be fulfilled by different set of tasks or user requirement can be changed at run-time. The areas of application may include Hospital Management Systems, Inventory Management Systems, game applications, Databases and Smart Searching Applications for Security Agencies at various levels and many others.

1.6 Structure

The thesis work is divided in chapters, a brief preamble of each is:

Chapter 2 gives a detailed account of conducted literature survey for the research to materialize, Chapter 3 Describes our proposed architecture of reasoning component, Chapter 4 Covers a case study to demonstrate the proposed architecture using goal model and HTN/PDDL (planning domain definition language) representation, Chapter 5 is about Evaluation of the implemented model, Chapter 6 contains of Concluding remarks and recommendations for possible Future Work.

Chapter 2

Literature Survey

2.1 Overview

The hot area of *self-adaptive systems* (SAS) has gained significant importance over the past few years owing to their adaptability to diverse scenarios. Many new issues in this field of SAS are identified and have been acknowledged. Currently the main focus of research is to provide run-time adaption to the software applications by using design time solutions. Although lot of work is done in this area and many issues are addressed but still requirement engineering of self-adaptive systems has not gained much attention which ultimately result in a gap between design-time and run-time adaptation activities. According to the main concept of self-adaptive systems, they should not only aware of their own requirements but also be able to monitor their end-users goals and preferences, operating context, domain conditions and etc.

This chapter gives overview of *self-adaptive systems*, modeling dimension of Self-adaptive systems, requirement engineering of self-adaptive systems and CARE Framework.

2.2 Self-Adaptive systems

With the disclosure of long lived mobile and distributed systems it become very difficult to manually manage such large and complex systems. This transform leads to the vision of *self-managed* systems which have a capability of *self-adaptation*, *self-configuration*, *self-monitoring*, *self-healing*, and *self-tuning* [2]. Basic aim of *self-healing* and *self-adaptation* is that the system should adapt or re-configure according to the new requirement specification or changes in its environment or report an exception [39].

Self-adaptive systems are the one which are able to alter their response at run-time according to changes in the system operating environment or changes in system\user requirements. There is a deprivation of consent among researchers and practitioners on the points of discrepancy among such software systems. These points of discrepancy are called *modeling dimensions* in [1] which focus on illustrative model of *modeling dimensions* for *self-adaptive systems* and classify them in four major groups i.e. Goals (targets which software system should achieve), changes (which are the causes of adaptation e.g. Change in environment, changes in requirements etc.), mechanisms (response of system towards change), effects (which are effects of adaptation).

2.2.1 Modeling Dimensions of Self-Adaptive Systems

The identified *modeling dimensions* of SAS are placed in four major categories in [1] as stated above. First category includes dimensions related to system goals. Category second includes dimensions related to the causes of *self-adaptation*. Category three includes the dimensions related to the mechanisms used to achieve *self-adaptation*. Finally fourth category includes dimensions associated with the final effects of *adaptation* on system.

2.2.1.1 Goals

The objectives that system should achieve are called goals [13] which are linked with life of system or with scenarios associated with the system. Goals may also be related to *self-adaptive* mechanism of software application or also to some infrastructure or Middleware framework which provide support to the application. Higher category of goal include following dimensions.

- *Evolution*: This *modeling dimension* represents the alteration of *goals* within the system and it range from static to dynamic. In static evolution changes do not occur at run-time whereas in dynamic evolution changes occur at run-time.
- *Flexibility*: This *modeling dimension* represents the flexibility of goals and is directly related to the level of precariousness related with the goal specification. It may span over three values: rigid, constrained and unconstrained.
- *Duration*: This *modeling dimension* is directly related with the validity of a goal during the systems complete life time. Duration of goal range from temporary to persistent. Temporary *goal* is viable for short medium or long

period of time while persistent goal should be valid throughout lifetime of system.

- *Multiplicity*: This *modeling dimension* is concerned with the goal count related to the *self-adaptive* mechanism of system. Software system may have single or multiple goals.
- *Dependency*: This *modeling dimension* is concerned with dependency or independency of goals on each other in case of multiple goals.

2.2.1.2 Change

Changes are source of *adaptation* e.g. change in user or system requirements, changes in environment in which system is deployed, results in self-adaptation of system. Changes are classified in following four terms based on their place [1].

- *Source*: This modeling dimension is related to the origin of change. A change can either be external for example changes in system environment or systems internal change.
- *Type*: This modeling dimension is concerned with the identity of change which can either be technological, functional or non-functional.
- *Frequency*: This modeling dimension identifies how often a specific change occurs.

2.2.1.3 Mechanisms

This group of dimensions is directly related to adaptation process as it analyzes the system reaction towards any change. This group of dimensions is related to the system reaction towards change. This means that they are actually related to the *adaptation* mechanism. All dimensions related with mechanisms group point towards the expected type of *self-adaptation*, *degree of autonomy* of self-adaptation of the system, How the *self-adaptation* of system can be controlled, and how it reacts to change. Following dimensions are related to this group:

- *Type*: This dimension ranges from parametric to structural.
- *Autonomy*: Is concerned with the level of external interference of human or system during adaptation.
- *Organization*: Is either centralized or decentralized.
- *Scope*: It range from local to global
- *Duration*: Can be short, medium or long term.

2.2.1.4 Effects

This group of dimension is related to the impact of adaptation on the system. This group include following dimensions.

- *Criticality*: This dimension is concerned with effect on system in case of failure of self-adaptation.
- *Predictability*: It means whether the effects of adaptation are predictable or not.
- *Overhead*: This dimension is associated with the effects of adaptation on the QoS (quality of services) of system.

2.3 Self-Management

In software engineering there are series of conferences and workshops which started in distributed systems community with *WOSS (Workshop on Self-Healing and Self-Managed Systems)* [3, 4] and *SEAMS (Software Engineering for Adaptive and Self-Managing Systems)* [5]. Although the work discussed in them has large contribution towards *self-management* but it has not solved the some very important issues to move towards comprehensive and integrated approach. So architecture based Approach is used in [2] because it has some benefits like Generality, Level of abstraction, Potential for scalability and Potential for an integrated approach. Many other researchers are also interested in component based architectural approach and have done lot of work. For instance, Schmerl and Garlan [7] give the description of the usage of architecture models to support *self-healing*, Oreizy provide the abstract of architectural approach in [6] which include evolution management and adaptation. Van der Hoek , Taylor and Dashofy introduced the concept of using architecture evolution manager to provide supportive infrastructure for *self-healing* and run-time adaptation [40]. Rosenblum, Medvidovic and Taylor presented language and its related development environment in [41] for architecture based development. Hussein and Gomaa illustrated the usage of reconfiguration of software dynamically and the patterns of reconfiguration for software products in [42].

2.3.1 Architectural Model for Self-Managed systems

Architectural Model for Self-Management is proposed in [2] based on robotics architecture because the first architecture proposed for self-management is very close to SPA architecture used in robotics. This close relationship exists because a *self-managed* system is actually autonomous system like robot. Both try to achieve their goal by their own without any human interaction. Robotics is based on three layer architecture of GAT [8]. So in [2] researchers tried to translate the three level robotic architectural model for *self-managed* systems. The three layers are Control: which is feedback control, Sequencing: which is plan execution and Deliberation: is related to planning. In case of robotics the lowest layer i.e. control layer consist of control loops, sensors and actuators whereas in the case of *self-managed* systems it consist of interconnected components and self-tuning algorithms responsible for application functionality and also for reporting the present status of component to upper layers. It also supports component creation, interconnection and its deletion. The important feature of this layer is that it automatically detects failure and report to upper layer when some situation occurs and the current configuration of components is not designed to deal with such situation. The middle layer i.e. sequencing layer react to changes reported from lower layers. When a new situation is given this layer perform the sequence of actions to tackle a changed situation. It can create new components, change interconnections of components. Upper layer i.e. Deliberation layer is responsible for producing change management plans as a response to the request from lower layer and due to the new goals introduction.

Architecture proposed in [2] is not implementation architecture but conceptual architecture. This reference architecture is used to shape the presentation of research problems suggested by the challenge of implementing *self-managed* systems. In this context at control layer researchers are mainly concerned with management at architecture level where system consists of interconnected components distributed over a network. A component consists of two types of services both provided and required and an externally visible state called mode [48]. *Mode* represents that whether a component is in active state or standby state.

To deal with complex components *self-management* will entail the online dynamic realization of operations. At change management level the main issue is to deal with distribution and de-centralization. In complex applications distribution raise issues of *latency, concurrency* and *partial failures*. To cope with distribution and failures there is a need of autonomy locally while maintaining global consistency. At goal management layer, goals should be precisely specified, so that they are readable by machines. High level goals are broken down into low level task that are processed by machine. This is actually an idea of goal oriented requirement engineering.

2.4 Requirement Engineering of Self-Adaptive Systems

In software development life cycle requirement engineering is the ground activity upon which the working of whole system to-be depends. Requirement engineering is itself a process which includes a sequence of activities i.e. Requirement elicitation, analysis, specification and validation. Requirements are not static entity but they change during whole software development life cycle and even during system working. At run-time requirements changes due to change in user needs, resources and change in environment. But it is difficult to find, reason and handle requirements at run-time for *self-adaptive* systems. Feather and Fickas work on requirement monitoring is a key contribution towards run-time requirements [17]. Continuous requirement monitoring is necessary because of the deviation of system behavior at run-time from requirement model which ultimately triggers the demand for system moderation. Such deviations need to be agree with the changing condition of environment so that the reasons can be identified and suitable adaptation is achieved. This is called monitoring and switching by Salifu in [18] where the system adapts automatically in order to satisfy its goals.

Previously Requirements engineering was entirely static, off-line activity, but it is run-time activity also. This idea is first proposed by Berry in [19]. Berry also identify following four-level model for engineering requirements posed by system at run-time. (Level 1) include traditional RE activities done by analyst. (Level2) include run-time adaptive requirements. (Level 3) include requirement engineering done by analyst to determine adaptation mechanism which actually enables the system to adapt. (Level 4) include adaptation requirements that are associated to specific adaptation solutions.

2.4.1 Goal Oriented Approach

Goal Oriented perspective is extensively used in variability modeling and also during early RE (requirement engineering) for *eliciting, specifying, analyzing, and documenting* the requirements. The concept of goal oriented modeling and analysis revolve around the term *goal* which represent stakeholder requirement from the system i.e. Functional and non-functional requirements. The high level user goal is decomposed into sub-goals using *AND/OR* decomposition method. The sub-goals are further decomposed into variety of tasks, the satisfaction of which results in the satisfaction of sub goals and ultimately users high level goal. The AND/OR decompositions provides a basis to reason about alternatives according to different user requirements.

TROPOS methodology for goal modeling is used by Penserini in [20] to model run-time changes in user needs and preferences. It involves BDI (Belief-Desire-Intention) agents, which may switch from one behavior to another depending upon environmental conditions and change in user needs. Goal alternatives in user requirement model simply map to software agent goal model in design and code artifacts which help in answering different questions related to run-time adaptation. Software requirements for self-adaptive systems represented as goal models may be plotted to BDI (Belief-Desire-Intention) architecture which result into agent-based design framework to capture adaptive requirements, which may be a progressive step towards agent oriented self-adaptive software systems.

Liaskos in [21] used requirements driven approach to address the problem of changing requirements by configuring software using goal-oriented approach. He modeled user's high level preferences as goal alternatives and then matched them with the system's configurations. So in this way he supports reasoning about goal models to achieve automatic system configurations. This approach i.e. goal based seems very useful to depict the behavior of autonomic elements.

Zhu in [22] used goal models to derive patterns of autonomic elements. To express different autonomic patterns goal oriented RE approach and attribute based architectural style is used. In the field of goal oriented requirement engineering Jureta [23] redefined the concepts of core requirement ontology. The core ontology is mainly based on goal oriented concepts and also on mentalistic notions which are called modalities.

The main use of this core ontology is that it help in understanding the given requirement problem in more precise way. But the concepts of this ontology are not sufficient to build complete self-adaptive software because they are not sufficient to deal with changing requirements aspect and also adaptive nature of requirements.

2.4.2 Requirements Monitoring Approaches

Although different goal oriented requirement approaches provide base for refinement of run-time requirements and architecture. KAOS goal modeling is used by Feather in [24] to monitor changing requirements and to reason run-time behavior of system, so that it can adapt dynamically at run-time using pre-defined adaptation methods.

Robinson in [25] used KAOS approach to propose requirement monitoring framework which was previously named REQMon and now it is called EEAT i.e. Event Engineering and Analysis Toolkit. This proposed framework actually monitor requirements related to web services. Robinson approach is mainly based on a strategy to identify requirement obstacles to monitor them and to do so requirement analyst should define a monitor that help in retrieving the needed data .In this context a version of object constraint language is used to specify monitors.

According to Salifu requirement monitoring is necessary for requirement variation and it help in composing the switching behaviors [26]. But Salifu's requirement monitoring approach is different from the approach used by Feather in [27]. As his approach also monitor problem variations besides the core requirements and also help in deep understanding of contextual variability. But in self-adaptive systems aspects his approach is limited to only scenarios.

2.5 Continuous Adaptive Requirements Engineering Framework

The existing RE approaches anticipate run-time changes at design-time, so they are unable to accommodate new or changed requirements posed by the end-users at run-time. In this context a novel framework is proposed called Continuous Adaptive Requirement Engineering framework. CARE is user and goal oriented RE framework that captures and analyzes user requirements at run-time [7].

The main idea behind CARE is that the system itself plays the role of analyst i.e.it perform RE activities at run-time to fulfill end user preferences and adapt itself to meet modifications in user goals and preferences.

To achieve this adaptation system automatically updates its knowledge about its operational environment and end user needs. The requirements captured by system at run-time are called service requests [5] in CARE which can be expressed in XML format. These Service requests consist of either new requirements or refined set of requirements expressed as *goals*, *quality constraints*, *preferences*, *priorities* etc. These service requests are provided as an input to reasoning component. The Reasoner performs three operations on incoming data (RRA). First it Evaluates the incoming RRA to determine which type of adaption (1 to 4) is required [6] before activating a planner. After evaluating the Plan activity activates the planner to generate task sequence and the selected plan is executed by Reasoner Adapt activity [7].

2.5.1 RE at Design-Time vs RE at Run-Time

CARE framework support two types of RE processes i.e. design-time and run-time RE processes which are dependent on each other. During RE at design time system analyst utilizes user needs and performs analysis on them to formulate requirement problem which in return help in formulating requirement specification. During RE at run-time the requirement activities are performed by SAS by involving end user in order meet requirement changes that are posed by user during system use.

2.5.1.1 RE at Design-Time

RE activities at design-time are performed by analyst. Analyst gathers requirements from stakeholders. Requirements gathered from stakeholder are expressed as *goals* and *quality constraints*. Goals may be *mandatory goals* or *optional goals*. The whole requirement problem is represented as goal model which consist of *nodes* i.e. goals and *edges* i.e. relations between goals. Solution of requirement consists of collection of tasks which are performed in any suitable sequence to achieve its high level goal.

RE at design time include a sequence of activities i.e. requirement elicitation, analysis, elaboration and specification. During elicitation phase requirements are gathered by analyst from stakeholders. These requirements are then formalized into goals, quality constraints and user preferences. Following the elicitation phase is analysis phase during which requirements for SAS are analyzed. Goals i.e. mandatory and optional goals are decomposed into sub goals and further into tasks. Soft goals are added in this phase to analyze the requirements that can influence other requirements or they can be influenced by each other.

The resulting requirement model is further extended in elaboration phase in which requirement problems are elaborated to describe system properties. Adaptive actions are specified as tasks to make sure that system behavior will meet them when operating in dynamic environment. In last phase i.e. specification analyst finds the set of tasks which satisfy mandatory goals. Specification is actually solution of requirement problem which is used by SAS at run-time to reason for adaptation.

2.5.1.2 RE at Run-time

In this phase the system plays the role of analyst. The user changing requirements, preferences and environmental changes are captured by system its self at run-time. Run-time requirements are called service requests which include user functional goals, preferences, context information (user or operational) etc. This concept of service request is introduced to manage requirements at run-time and is expressed in XML format to make them machine readable which are called run-time requirement artifacts (RRA) at this stage.

As stated above RE activities at run-time are performed by SAS itself. Following are CAREs main RE activities.

Service Request Acquisition: This activity is carried out to gather service requests from end user. After acquiring RRA from user they are compared with existing requirement specification to find which operation to be performed. These operations allow Self adaptive system to reason for refining its requirements. For example add new goal or task, substitute existing goal or task.

Service Lookup: Service lookup is performed to look suitable services according to users requirement, in a pool of services of the SAS or using a *web service* search, e.g. *Woogle or Seekda*. Services are find by comparing users keywords with service descriptions and a list of suitable services is displayed to user according to his preferences.

Service Selection: During service selection user is involved for the selection of service. List of available services is displayed to the user for its selection. After user selection and confirmation RRA is again refined and details are added about the selected services.

Update Specification: This activity is initiated to update the specification PO after the RRA is filled with all user inputs and information from monitors. Update is done through the addition of new requirements or refinement of existing ones.

2.6 Summary

Existing RE approaches work well where the requirements of the system remain static. Although these approaches handle run-time requirements at design time but they are unable to handle changes in requirements problem posed by the user during system working. To cater this problem of run-time adaptivity a novel framework was proposed named as CARE framework which tackle user requirements both at design-time and run-time. The conceptual architecture of CARE was already proposed which consist of various components interacting with each other to perform run-time adaptation.

In this thesis work we are working on reasoning aspects of CARE framework by exploring goal modeling and AI Planning techniques. These techniques are integrated in care framework to perform run-time reasoning of requirements. The subsequent chapter explains our proposed architecture of CARE reasoning framework and brief description of goal modeling and AI planning techniques which we have used in our work to perform run-time reasoning.

Chapter 3

Proposed Architecture

3.1 Proposed Process

Fig 3.1 shows the modified architecture of CARE reasoning framework integrating goal modeling and automated planning techniques [31] [12]. The requirements are represented as goal model [9] which not only represents the functional requirements but also the users preferences and non- functional requirements. In CARE these goal trees are considered as a pool of *adaptive requirements* which are directly translated into planning action theory [10] by using HTN semantics. This action theory represents that how user goals can be achieved in most suitable way under given conditions. The architecture of CARE reasoning component is self-adaptive where different components automatically interact with each other to support run-time reasoning of requirements. It consists of following set of agents:

- *User agent*
- *Planning agent*
- *Lookup agent*
- *Update agent*

The requirements are captured from the user through the user agent which are called run-time requirements artifacts (RRA). These RRA consist of various requirement elements e.g. *User Hard and Soft Goals*(G, SG), *Preferences*(P), *Quality Criteria* (Q) and also contain data coming from *Monitors* that senses the changes in *Environmental Conditions*(E).

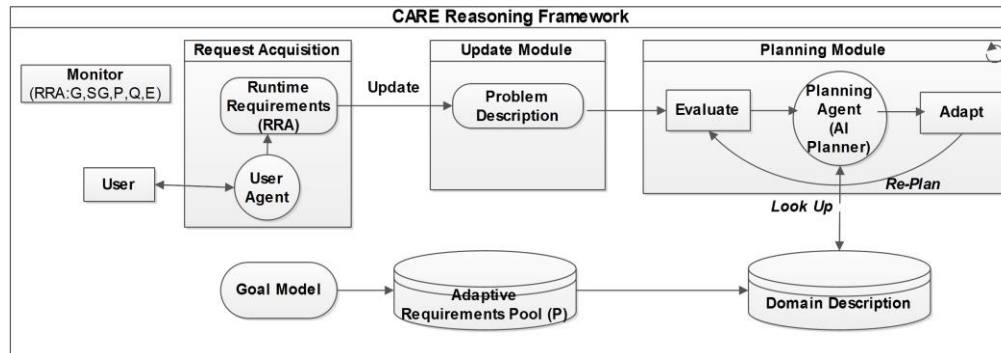


Figure 3.1: Modified CARE Reasoning Framework

These users RRA are convertible into complete problem descriptions in which initial conditions contain the value of monitored variables, goal specification describe user goals and preference specification describe quantitative prioritization of preference goals [10]. These problem descriptions are passed to the planning agent who calls the AI Planner. As soon as the planner gets activated the Lookup agent starts searching (using A* Algorithm) for the best possible plan (sequence of task) in domain description that satisfy user goals defined in given problem description. The resulting plan is displayed to the user again through the user agent. Re-planning is required if the prescribed plan is not executed properly and new plan needs to be generated with different initial conditions. The updater agent is responsible for updating the problem description according to the new requirements posed by user through user agent.

3.2 CARE Reasoning Methodology

Our CARE reasoning methodology consists of four main practices:

1. *Requirements Engineering*

Input: Textual requirements from user/stakeholder

Output: Requirement goal model

2. *Reasoning Model*

Input: Requirement goal model

Output: Reasoning model, i.e. problem and domain specification

3. *Runtime Architecture Modelling*

Input: Reasoning model

Output: Architectural model, i.e. The run-time model used to represent and adapt the system at run-time.

4. *Deployment*

Input: Reasoning model and architectural model

Output: Running CARE Reasoning system

3.3 Goal Modeling and HTNs

In requirement engineering (RE) domain, goal modeling techniques have acquired major attention as it bridges the gap between users goals and the means to achieve these goals i.e. actions, tasks, plans. But goal based modeling frameworks [11-12] which are in use handle goals as *Mandatory Requirements* that must be satisfied by any suggested solution. Such modeling frameworks are unable accommodate the preference requirements that might be presented by the user. So a framework [9] is introduced for both specifying *Preference Requirements* and *Priorities* among them, and also for using them to select specifications that fulfill the *Mandatory Requirements* while best satisfying the *Preference Requirements* and *Priorities*.

This idea started with goal model which express alternative sets of low-level tasks and operations that can fulfill high-level stakeholder requirements, known as goals. These low-level tasks/operations which fulfill high-level goals are known as tasks. Two types of user goals are known i.e. *Mandatory goals* and *Preference goals*. Former should be satisfied in any case while later change according to the priorities of user. Quantitative prioritization of Preference goals is used for evaluating alternative ways to achieve *Mandatory goals*. In order to generate plans\solutions a preference based planner is used to find alternatives that satisfy a given set of *Mandatory* and *Preferred requirements* [9].

HTNs (Hierarchical Task Network) equip us to handle mandatory goals. The benefit of HTN to reason about goal model enhances the performance of

the planner significantly, especially when its compared with the earlier attempts to use non-hierarchical planning. Tasks are organized in a hierarchical order in HTN and are reduced recursively into other subtasks. HTN domain consist of *Operators* and *Methods* which narrate possible means to achieve goals whereas HTN problem specification consists of a list of *Predicates* and High level *Tasks* that should be fulfilled. HTN based planner first reads domain and problem specification and then recursively performs HTN *Tasks* to achieve high level *Goal*.

3.4 Parsers and AI Planners

3.4.1 Planning4J and PDDL4J

Planning4j is java based API used in AI planning [32].It provides convenience of using PDDL files in different planning applications. As we know that PDDL is language which is hard to handle and cannot be used directly in application. It require some intermediate parser or API to which can translate problem and domain file of PDDL into some other language used in corresponding application and also call a planner to run PDDL files to generate task sequence. In this context planning4j is very useful java based API which not only convert problem and domain files in specific way but also help in calling planner to run that files and translate PDDL files according to different planner.

Planning4J consist of 3 basic components which are problem providers, domain providers and planners, which run problem and domain files. In PDDL, problem and domains can be specified in variety of possible ways but different planner implementations need different ways. So for ease of use planning4j introduce providers for both domain and problem. Provider is actually an object which present problem and domain files in particular way. In planning4j (version1.1) there are 4 types of provider and their respective classes are called XXXX Problem provider and XXX Domain provider. These providers are: PDDL Object, PDDL string, PDDL File and IPDDL Writer.

Like Planning4J ,PDDL4J is another open source library which support java implementation of PDDL based planners[33].PDDL4J library contain parser of PDDL which also parse PDDL problem and domain files into java but this parser is configured to accept only specific requirements of PDDL like strips, typing,disjunctive and negative preconditions, conditional effects etc.

3.4.2 ANTLR V.3

In our work we are using ANTLR which is another tool for language recognition. It is powerful parser generator software that can be used to read process and translate text and binary files[34].It is both used for research and also in industry to build all kind of languages, tools and frameworks. ANTLR generate a parser for any language from a grammar of that language. The generated parser automatically builds parse trees which are actually data structures of that language representing how grammar matches the input. ANTLR also generate tree walkers that can be used to visit the nodes of trees to generate application specific code.

As sated above PDDL is a language which cannot be used directly in planning applications. Its problem and domain files should be translated into some other known language i.e.xml, java or.NET etc.to use in planning applications. So we are using ANTLR parser to parse problem and domain file of PDDL to java file which are then fed to JSHOP2 planner to generate task sequence [35]. The JSHOP2 compatible PDDL grammar is fed to ANTLR which compile domain description and problem description written in JSHOP2 grammar to java code. The resulting java code can be used in any self-adaptive application to handle run-time requirements.

3.4.3 AI planners

As we know that planning problems are represented in PDDL, STRIPS or HTNs which are processed by AI planners to generate solution i.e. task sequence. The existing AI planners can be divided into following three main categories:

3.4.3.1 Domain Specific Planners:

These planners work only in single problem domain and are built from scratch for each specific problem. So that efficiency of planning process can be increased. Following are commonly used domain specific planners:

- a. Remote Agent (Muscettola et al. 1998)
- b. Bridge Baron (Smith, Nau, Throop 1998)
- c. ICAPS (Nau, Regli, Gupta 1995)
- d. Mars Rover (Dias, Lemai, Muscettola 2003)

3.4.3.2 Fully Automated Domain-Independent Planners:

These planners include all classical planning systems. These general purpose planners can be reused with many domains without much effort just by abstracting the basic planning mechanism.

3.4.3.3 Hand-Tailorable Domain-Independent Planners:

These are usually general purpose planners also called domain configurable planners but as compared to automated planners they provide rich domain description language in which input to planner also includes advice to the planner for how to search for a plan to achieve a goal/task. Whereas in automated planners domain description only contain planning operators. In other words hand tailor able planners are such kind of domain independent planners which give option to their users to Specify domain-specific advice in domain description. Some common domain independent planners based on HTNs are:

- a. SHOP
- b. SHOP2
- c. JSHOP and JSHOP2
- d. HTN PLAN-P planner

Why JSHOP2? In our self-adaptive application we are using JSHOP2 planner [35] which is java implementation of SHOP2 [36]. We are using JSHOP2 because it is an efficient planner and its basic functionality is based on planning formalism called hierarchical task network planning [16] [35]. In most of the automated planning system planners have to try different possibilities before finding a workable plan because they do a trial-and error search of a large space of possible solutions. But HTN planners perform this same search by applying HTN methods which describe how to decompose *tasks* into *subtasks* to create a network of planning problem [37]. This hierarchical network of tasks divided into subtask can be efficiently searched/ solved by planner to generate task sequence.

3.5 Summary

In this chapter we presented our modified architecture for implementing CARE Reasoning framework. Our main focus is to design a system/architecture empowered enough to interactively take input from user and sense the operating environment; and consequently able to assess and reason about both at run-time to generate an intelligent and efficient solution/line of action. This feature of end user involvement at run-time to reason about requirements and providing plans for changed requirements at run-time by using available services distinguish CARE Reasoning framework from the state of the art approaches.

Chapter 4

CARE Reasoning Application

4.1 Application Scenario

Consider a human actor who is using INSTA PLANNER application which is installed on his Smartphone. The application act as a virtual secretary to the user which helps him in achieving his daily goals in most suitable and convenient way. The application covers basic daily tasks of various professionals i.e. Lecturer, Surgeon and Businessman with the flexibility/adaptability of incorporating his preferred selections and forced constraints met during plan execution. For instance, according to his preference of reaching his work place cheaply with no sense of urgency the application suggests him to move via train after evaluating the weighted preference against cost of executing the intermediate tasks in all possible cases as per defined Goal Model (see Figure 4.2,4.3).Application also remind him to pack different things before leaving and also remind him that he has to go to different places to complete his routine activities i.e. *go to gym, read news, write novel* etc. Moreover INSTA PLANNER also gives him suggestions for various activities on holiday based on different day time i.e. morning, evening and afternoon.

Identification of set of Goals, set of AND/OR decomposed tasks along with their pre-requisite conditions to meet the goals, predefined methods encapsulating in order various tasks for meeting minor goals, system ground state conditions and in addition a pool of adaptive requirements rendering Planning problem to become a Problem Set. The goals/tasks are also differentiated as human and machine tasks for example *Pack Bag, Carry Wallet, Pack Laptop* are the tasks performed by user but they are suggested as a reminder by application. System notifies *Faulty ATM Machine* and suggests user to *Use Cheque to Draw Cash*.Preference goals are also added for example

Reaching Urgent, Enjoying Route, Cost Effective, etc by assigning weighted metrics to each preference and evaluating the core heuristic function with the actions accumulated costs.

4.2 Implementation of CARE Reasoning Framework

The prototype application based on the above mentioned scenario in order to extend the desired features of adaptability and handling of run-time requirements entailed a Goal Model so designed that encompassed a few possible eventualities and recursive corrective actions. The graphically represented goal model is then mapped to its equivalent PDDL specification which is then fed to the planner to generate plans.

The following sections explain the implementation of CARE reasoning application starting from goal model to its PDDL specification.

4.2.1 Techniques of Goal Modeling

There are different techniques of goal modeling which can be used to represent the above mentioned application scenario. These techniques include:

1. i^*
2. KAOS

These modeling techniques can be used at different stages of requirement engineering i.e. requirement elicitation, requirement specification etc. i^* is used during early stages of RE when requirements are not clear enough and goals are not well defined. It focuses on understanding enterprise goals and how they effect on the behaviors of actors .Whereas KAOS approach focus on relating the functional and non-functional requirements to the enterprise goals because it assume the sufficient knowledge about the current organizational state.

4.2.1.1 i^* and TROPOS

i^* modeling framework is modeling language used during early phases of RE for modeling and reasoning about heterogeneous actors with different goals and task in different problem domain.

There are several modeling techniques which are based on i^* framework and support modeling in i^* i.e. TROPOS. TROPOS is both agent and goal oriented software engineering methodology that cover the complete software development process from requirement analysis to implementation. According to TROPOS methodology the notion of agent and other notion like actors, goals and tasks are used in all phases of development process. TROPOS also support modeling of self-adaptive systems to design and execute requirement driven self-adaptive systems. **Tools:** There are number of prototype tools which are used for TROPOS methodology shown in Figure 4.1.

Tool	Description
Si* Tool	It is developed by University of Trento. It is one of the secure TROPOS tool which support risk planning and reasoning.
TAOM4E	It is developed and maintained by FBK IRST in Trento, Italy. It is also a tool for agent oriented modeling based on TROPOS.
GR-Tool	It is reasoning tool developed by University of Trento.
T-Tool	It is Formal TROPOS tool developed by University of Trento.
DW-Tool	It is tool for Data Warehouse design designed by University of Trento.
OpenOME	It is General purpose tool based on i^* and is developed by - University of Toronto.

Figure 4.1: Modelling Tools

4.2.1.2 KAOS

KAOS is GOAL driven requirement engineering methodology. It enables analysts and requirement engineers in building requirement models which are also called KAOS models. It help in Formal Modeling of both functional and non-functional requirements in terms of goals, events, actions etc.[38]. In KAOS goal model the term requirement and expectations are used separately. Requirements are tasks which should be achieved by software agent directly whereas expectations are tasks which have to be attained by the agent which is actually a part of system environment.

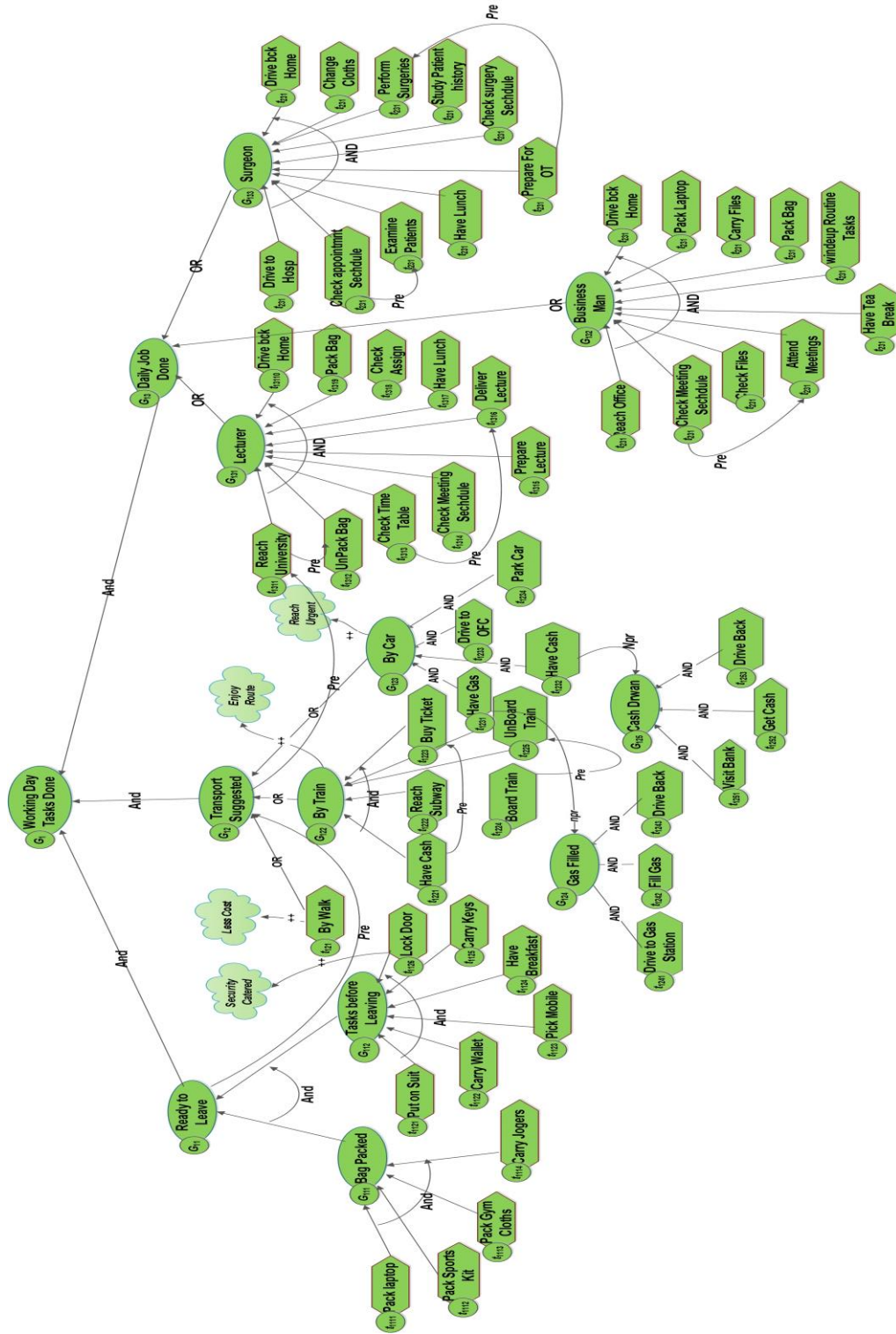


Figure 4.2: Working Day Module

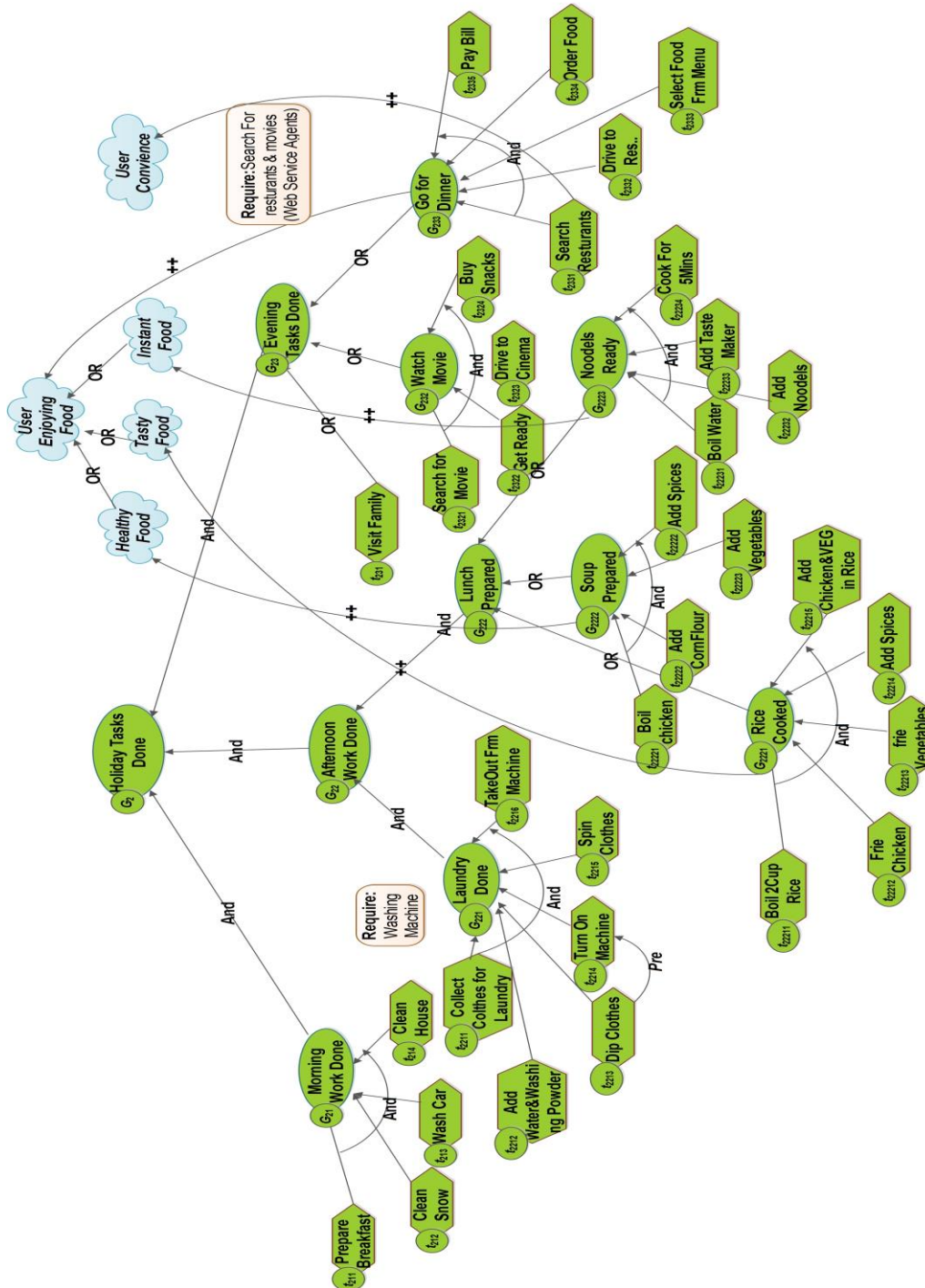


Figure 4.3: Holiday Module

4.2.1.3 Goal Models

(Figure 4.1 and 4.2) show the goal model of above mentioned INSTA PLANNER application scenario. The goal model of application is divided into two main modules i.e. working day module and holiday module. It is actually a decomposition model i.e. AND/OR decomposition. This model shows the alternative ways by which application suggest a user to achieve his weekly/daily tasks including doing job work, suggesting suitable transport, pack bag and do extra activities on holiday like washing clothes, cleaning home, go to gym, visit relatives etc. Application also suggests the most convenient route to the destination according to the requirement of user. Information about the route is a domain property. This goal model consists of goals and tasks. Ovals in the diagrams represent the goals which user wants to achieve and the hexagonal shapes represent the tasks which actor performs to fulfill goals. Thus bag packed is an example of goal and pack gym cloths, pack laptop etc are examples of tasks performed by the user to achieve goal.

An arrow with keyword *Pre* is drawn from one *goal/task* (origin) towards another *goal/task* (target) is precedence link which means that target cannot be started until the origin is fulfilled. For example user cannot drive the car if gas is not filled. This is positive precedence. In negative precedence the target cannot begin if origin is satisfied. Precedence links are used to represent the constraints which are not directly the desire of any user but they are the desire of domain for example cash is required for buying ticket. Goal model in (Figure 4.2, 4.3) is a subset of *i** Strategic Rationale Diagram but PRECEDENCE LINKS are additional to the main concept of *i**. This concept of precedence links taken from Goal model in [9].

All the above mentioned goals that are discussed up till now are mandatory goals that need to be fulfilled for the root goal satisfaction. Another type is preference goals which are the goals whose fulfillment is desired but not compulsory because there non fulfillment dose not result in the dissatisfaction of root goal. Preference goals may be Hard or Soft goals. The goals shown as cloud like ovals are soft goals and there is no specific criterion to decide that whether these goals are satisfied or not. For example Enjoy Route is soft goal and it is difficult to decide that how much user enjoyed the suggested route.

A plan [9] is devised for root goal satisfaction. Plans are suggested by planner based on the preferences of user. For example following sequence is a plan when user wants to reach urgent and also does not have cash in wallet and gas in car : $t_{1251}, t_{1252}, t_{1253}, t_{1241}, t_{1242}, t_{1243}, t_{1233}, t_{1234}$.

4.3 Mapping Goal Model to PDDL using HTN

This section explains how the above mentioned User Goals, Sub Goals, Preferences and Tasks are translated to HTN and JSHOP2 compatible PDDL specifications to solve the planning problem and how action parameters and domain predicates help in the richer representation of domain and its states:

4.3.1 Planning Domain

Domain description is the knowledge base we prepare for the planner in order to enable it to solve the problems presented to it with regards to this specific domain. The domain description if done to the minutest details, catering for all the possible actions that might be involved in solving the possible problems enhances the planner's response in giving valid solutions or task lists. Domain is composed of various predicates, operators, axioms and methods. JSHOP2 does not operate on standard PDDL, but a variant of it defined in LISP and dictated separately in its own grammar. Therefore JSHOP2 compatible translation of prior mentioned scenario through its equivalent logical model has been accomplished keeping in view the possible requirements that might be brought forth at run-time.

For the Transport Module of the application the modes of transport were defined in a single category i.e. (Via ?Mode). Similarly all the locations as (Present-At ?Loc) and so on. Reaching the work place via each selected mode has been represented by means of methods, which are further subdivided into a set of ordered primitive tasks for each. Since the selection of mode of transportation had to be made upon evaluation of user based preferences, axiom of (:- Mode-Sel ?Mode) is used to reason/evaluate and chose the preferred method of reaching the destination work place. The pseudo codes of domain descriptions for different scenarios of reasoning application are shown in Figure 4.4, 4.5, 4.6:

4.3.2 Planning Problem

Planning problem is the precise description of planning problem at hand, which is expressed after declaring the Ground State of the domain with the help of domain predicates. Problem domain also includes the identification of various object types i.e. actors in our planning problem along with Problem specific knowledge. For example transport module of the application has identified three different modes for reaching work place, each attributed with certain preference depending upon the user specification. The corresponding problem for above mentioned transportation domain includes the declaration of Objects i.e. Transport (walk, car, train), Locations (Home, Gas-Station, Bank, WorkPlace, Subway-Station-A, Subway-Station-B) and their user defined metric preferences for each.

Domain Axioms:
Is-Urgent(X) =>Mode-Sel(X)
Is-Enjoyable(X) => Mode-Sel(X)
Cost-eff(X) =>Mode-Sel(X)

Methods & Operators:

Operator: *Walk (Loc-from,Loc-to)*
Pre-O =Is-At (Loc-from)
Eff-O =Is-At (Loc-to)
Operator: *Drive (Mode-car, Loc-from, Loc-to)*
Pre-O =Mode-Sel(Mode-car)^Car-At(Loc- from)^Have-Gas(Mode-Sel)
Eff-O =Car-At(Loc-to) ^Is-At(Loc-to)

Operator: *Ride (Mode-train, Loc-from, Loc-to)*
Pre-O =Mode-Sel(Mode-train) ^Is-At(Loc-from) ^ Have-ticket(Mode-train)
Eff-O =Is-At(Loc-to)

Method: *Via-Car(Mode-Car, Loc-from, Loc-to)*
Pre-M = Car-At(Loc-to) ^Is-At(Loc-from) ^Need-Gas(Mode-Car)
Tasks-M={Unpark(Mode-Car), Drive(Mode-Car, Home, Gas-Station), Fill-Gas(Mode-Car, Cash), Pay(Cash, Gas-Station), Drive(Mode-Car, Gas-Station, Office), Park(Mode-Car)}

.....

Figure 4.4: Pesudo Code for Planning Domain (Transportation Scenario)

Methods & Operators:

Operator: *PickUp (Loc-now, Loc-goal)*
Pre-O =Obj-At (Loc-now)
Eff-O =Obj-At (Loc-goal)
Operator: *PutIn (Loc-now, Loc-goal)*
Pre-O =Obj-At (Loc-now)
Eff-O =Obj-At (Loc-goal)

Method: *(Obj-Place Obj-c Loc-goal)*
Pre-M = ((Obj-at Obj-c Loc-now))
Tasks-M= {{{PickUp}Obj-c Loc-now} (PutIn Loc-goal Obj-c)}}

.....

Figure 4.5: Pesudo Code for Planning Domain (Packing Bag Scenario)

Methods & Operators:

Operator: *Reach (Work-place)*
Pre-O =*Is-At (Home)*
Eff-O =*Is-At (Work-place)*

Operator: *TakeOut-fromBag (item1, item2)*
Pre-O =*Is-At (Work-place)*
Eff-O =*Bag (Empty)*

Method: (*perform-job-tasks user*)

Pre-M = (*(works-at (work-place, user))*
 (*commitment-chart-of (user, schedule)*)
 (*job-task-of (user)*)

Tasks-M= (*{Reach (Work-place)*
 (*Take-out-from-Bag (item1,item2)*)
 (*Check (Schedule)*)
 (*Prepare-for (Schedule)*)
 (*Perform (JobTasks)*}

.....

Figure 4.6: Pseudo Code for Planning Domain (Daily Job Tasks)

Planning Problem:

Init:
{Is-Urgent(Car),Is-Enjoyable(Train),Is-Cost-eff(walk),
Need-Gas(Car),Have-Cash(Cash),....}

Goal:
{Reach-Work-Place(Pref-U,Pref-E,Pref-C)}

Figure 4.7: Pseudocode for User Planning Problem 1

Fig 4.7, 4.8, 4.9 show pseudocodes of problem description for transportation scenario and some other problem scenarios posed by the user.

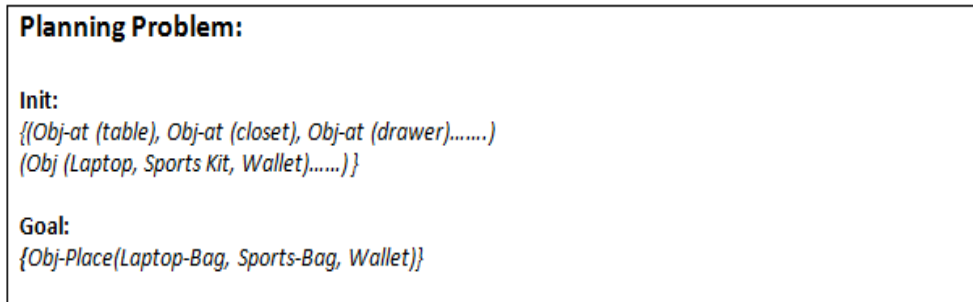


Figure 4.8: Pesudocode for User Planning Problem 2

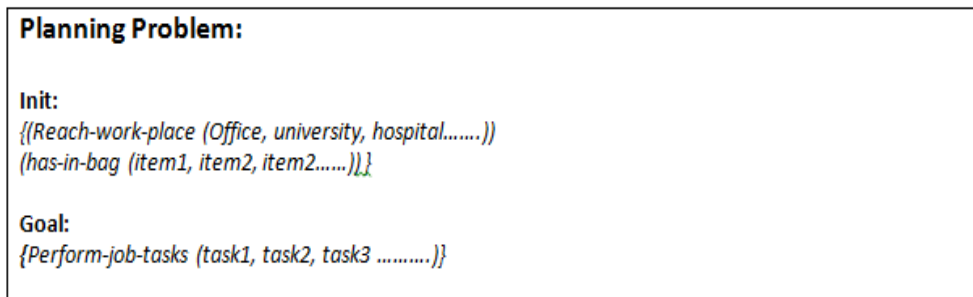


Figure 4.9: Pesudocode for User Planning Problem 3

4.4 Development Environment of CARE Reasoning Framework

Prototype application named i*INSTA PLANNER has been developed to validate proposed architecture of CARE reasoning framework. Application has been developed using java Net Beans IDE, MySql and JSHOP2 AI-Planner which is java version of SHOP2.i*INSTA PLANNER is AI-Planner based self-adaptive application that incorporate online Web Services, offline MySql database, Virtual Sensors and generates plans to achieve daily goals of user based on data coming from virtual context sensors, web Services and user preferences.

In the following, implementation of different modules of application is explained with the help of algorithms. (See Fig 4.10, 4.11, 4.12, 4.13)

Application gets input from the user through UI and generates plans suggesting user the future course of action to be adopted based on his selected preferences. Following Screen shots show the implementation of application w.r.t to above goal models. (See Figure 4.14, 4.15, 4.16, 4.17, 4.18)


```

GeneratePlanForGettingReady()
    ProblemSpecifications -> GenerateProblemSpecification ();
    Invoke PlanningAgent (ProblemSpecification);
    Plan -> SearchPlan(DomainDescription);
    return Plan;

```

Figure 4.10: Algorithm for Getting Ready Module

```

GeneratePlanForTransportation()
    PlanGenerationMode -> GetPlanGenerationModeFromUser();
    if PlanGenerationMode = AUTO
        UserPreferences -> UserPreferencesFromConextSensing()
    else
        UserPreferences -> UserPreferencesForTransportation();
    ProblemSpecifications -> GenerateProblemSpecification (UserPreferences);
    Invoke PlanningAgent (ProblemSpecification);
    Plan -> SearchPlan(DomainDescription);
    return Plan;

UserPreferencesForTransportation()
    ModeOfTransportation -> GetModeOfTransportation();
    AdditionalInfo -> GetAdditionalInfo(ModeOfTransportation);
    UserRRA -> GetUserRRA(ModeOfTransportation,AdditonallInfo)
    return UserRRA;

UserPreferencesFromConextSensing()
    WeatherCondition -> WeatherWebservice()
    FuelLevel -> FuelMointering(FuelContextSensor)
    CashStatus -> CashMointering(CashContextSensor)
    AtmWorkingStatus -> PullInfoFromBank(ATMService);
    UserRRA -> GetUserRRA(WeatherCondition,FuelLevel,
        CashStatus,AtmWorkingStatus);
    return UserRRA;

```

Figure 4.11: Algorithm for Transportation Module

```
GeneratePlanForHolidays()
  SystemTime -> GetSystemCurrentTime();
  TimeOfDay -> GetTimeOfDay(SystemTime)
  if TimeOfDay = Morning
    ProblemSpecification -> GenerateProblemSpecifications(TimeOfDay)
  else if TimeOfDay = Afternoon
    Suggest Options For Afternoon Tasks
    if SelectedOption = Laundry
      ProblemSpecification -> GenerateProblemSpecifications (SelectedOption);
    else SelectedOption = Prepare Lunch
      UserPreferences -> UserPreferencesForLunch();
      ProblemSpecifications -> GenerateProblemSpecifications(UserPreferences);
    else if TimeOfDay = Evening
      Suggest Options For Evening Tasks
      if SelectedOption = VisitRelative
        ProblemSpecification -> GenerateProblemSpecifications (SelectedOption);
      else if SelectedOption = WatchMovie
        ProblemSpecifications -> GenerateProblemSpecifications(SelectedOption);
        ReserverSeatForWatchingMovie(); // System Action
      else if SelectedOption = GoForDinner
        ProblemSpecifications -> GenerateProblemSpecifications(SelectedOption);
        MakeResturantReservationForGoingForDinner(); //System Action
  Invoke PlanningAgent (ProblemSpecification);
  Plan -> SearchPlan(DomainDescription);
  Return Plan;
```

Figure 4.12: Algorithm for Holiday Module

```
ReserverSeatForWatchingMovie ()
    City -> GetCity();
    Cinema -> SearchCinema(City);
    Movie -> SearchMovie(City,Cinema);
    ShowTime -> CheckShowTime (City,Cinema,Movie);
    IsAvailable -> CheckAvailableSeats(City,Cinema,Movie,ShowTime)
    if IsAvailable = True
        MakeReservationofSeat(City,Cinema,Movie,Showtime)
        SendConfirmationEmailToUser(UserEmail)
    else
        Suggest User to Go for Dinner
MakeResturantReservationForGoingForDinner()
    FoodOption -> GetFoodOption();
    City -> GetCity();
    Restaurant -> Search Restaurant(FoodOption,City);
    TimeForReservation -> GetTimeForReservationFromUser()
    IsAvailable -> CheckAvailability(FoodOption,City,Resturant,TimeForReservation)
    if IsAvailable = True
        MakeReservationOfTable(FoodOption,City,Resturant,TimeForReservation)
        SendConfirmationEmailToUser(UserEmail)
    else
        Suggest User to Watch Movie
```

Figure 4.13: Algorithm for Holiday Actions Module

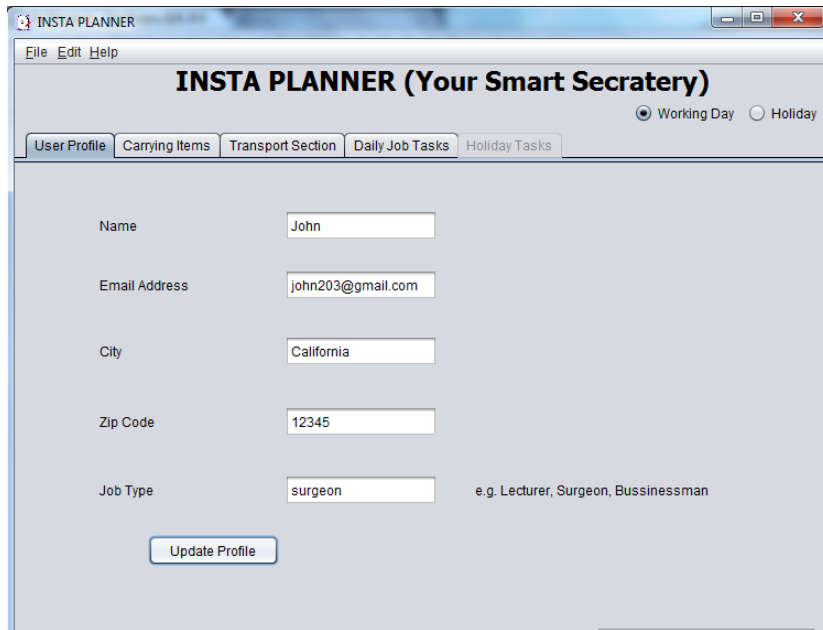


Figure 4.14: User Profile

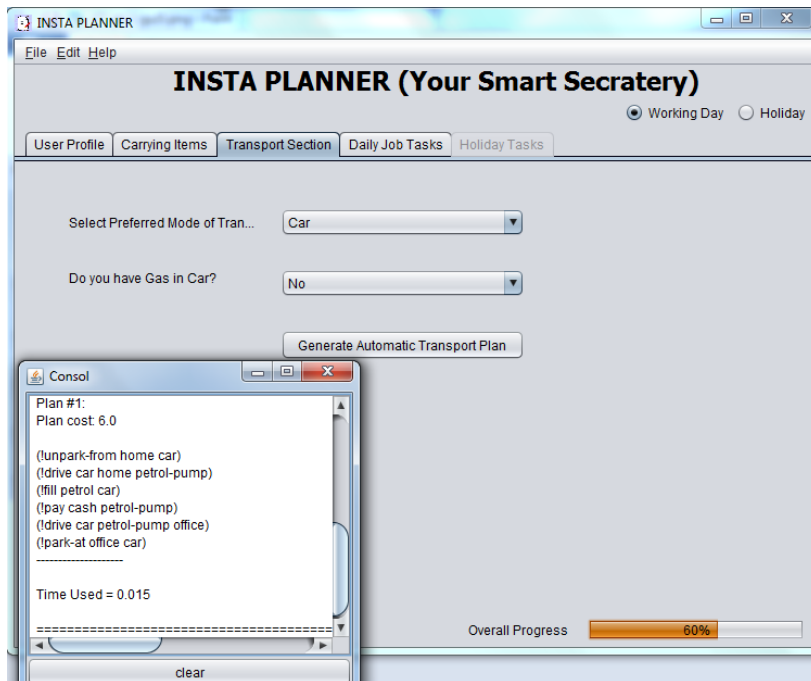


Figure 4.15: Transportation Module 1

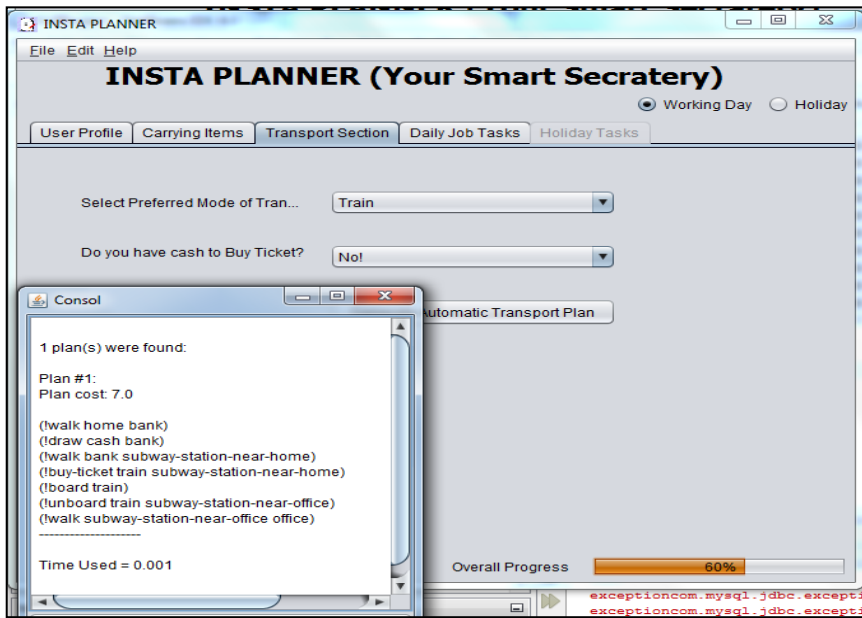


Figure 4.16: Transportation Module 2

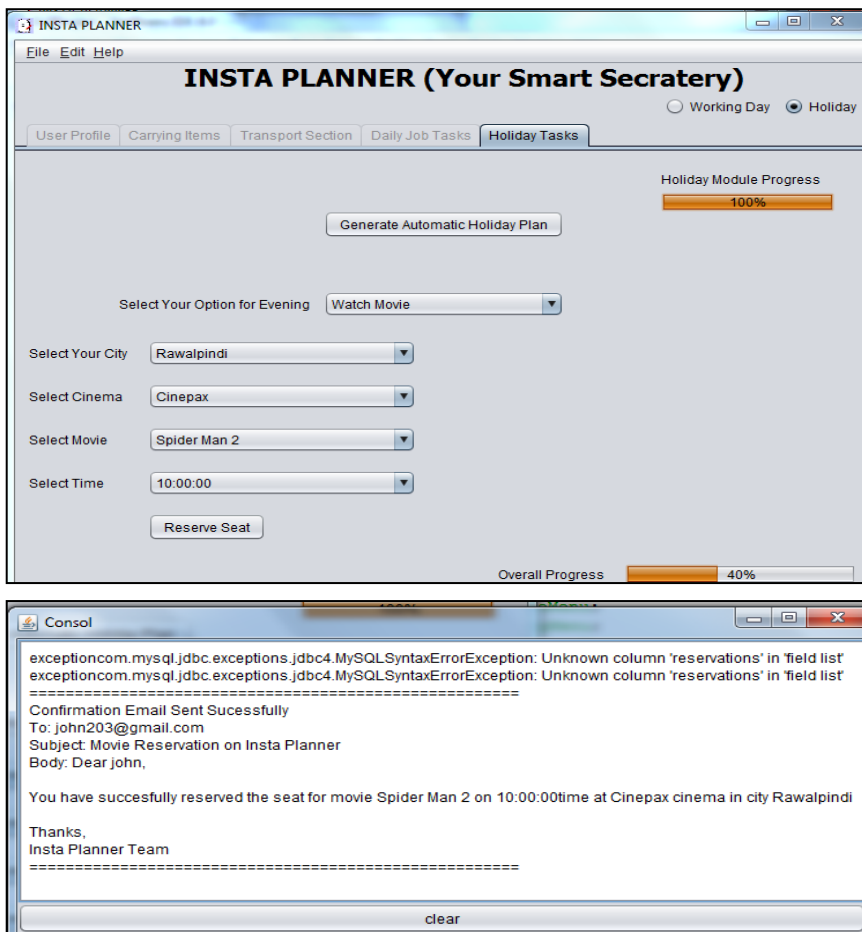


Figure 4.17: Holiday Actions Module

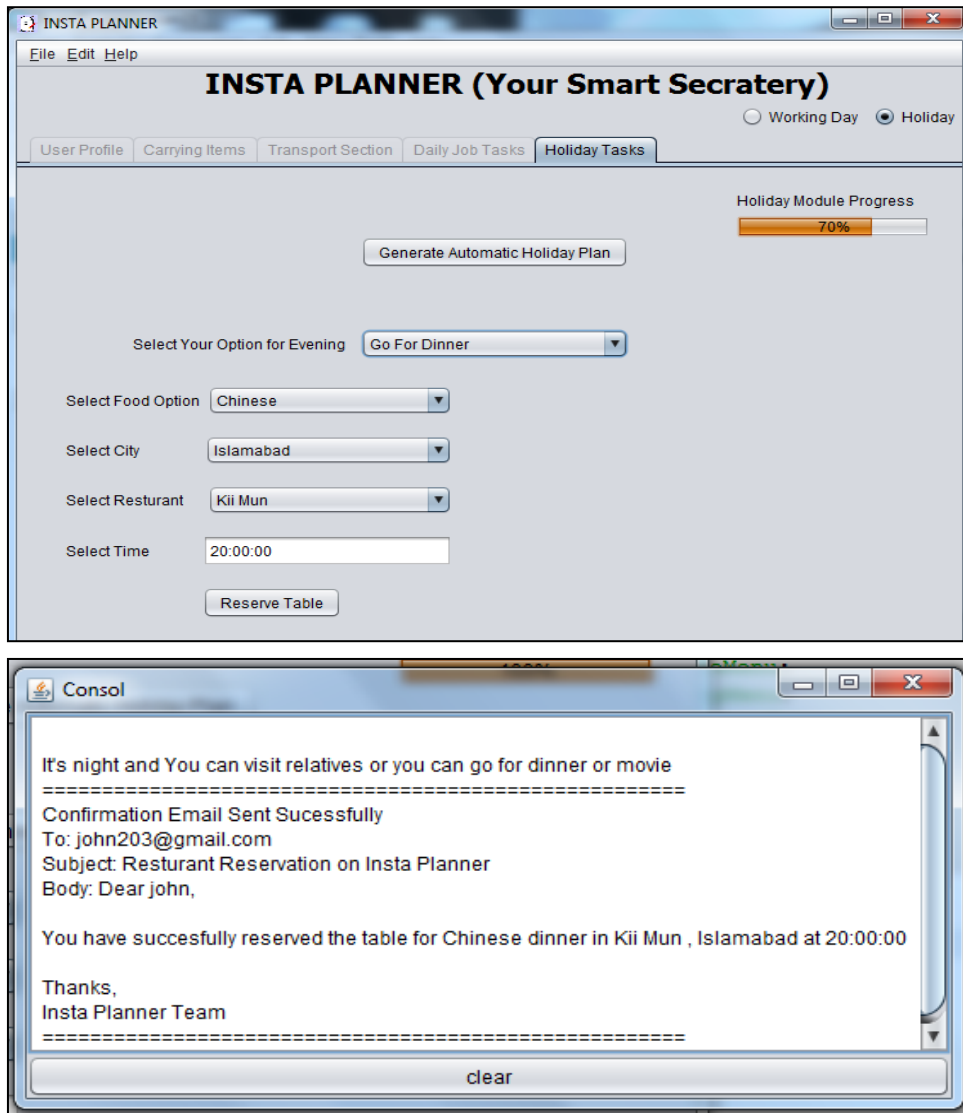


Figure 4.18: Holiday Actions Module

Chapter 5

Evaluation

In 1998 , Drew McDermott organized first international planning competition (IPC)[49,50,51,52] to evaluate AI planners in which only five planners have competed but many more planners competed in succeeding events of IPC which stimulated increase in planners performance. This results in improvements of planning system not only in terms of time to generate task sequence (plans) but also in terms of domain models . Fourth IPC results in the enrichment of domain models with derived predicates. Fifth IPC results in addition of soft goals in the model of planning problem.

In accordance with the results of International Planning Competition it was observed there are mainly three dimensions that are vital in the evaluation of any planning system/application:

1. Time taken by the planner to generate plans.
2. The number of problems presented to the planner.
3. Quality of plans generated by the planner.

5.1 Evaluation of Reasoning Framework

Proposed CARE Reasoning framework is evaluated on basis of two parameters:

1. Time taken by the planner to generate plans.
2. The number of adaptation scenarios supported by application.

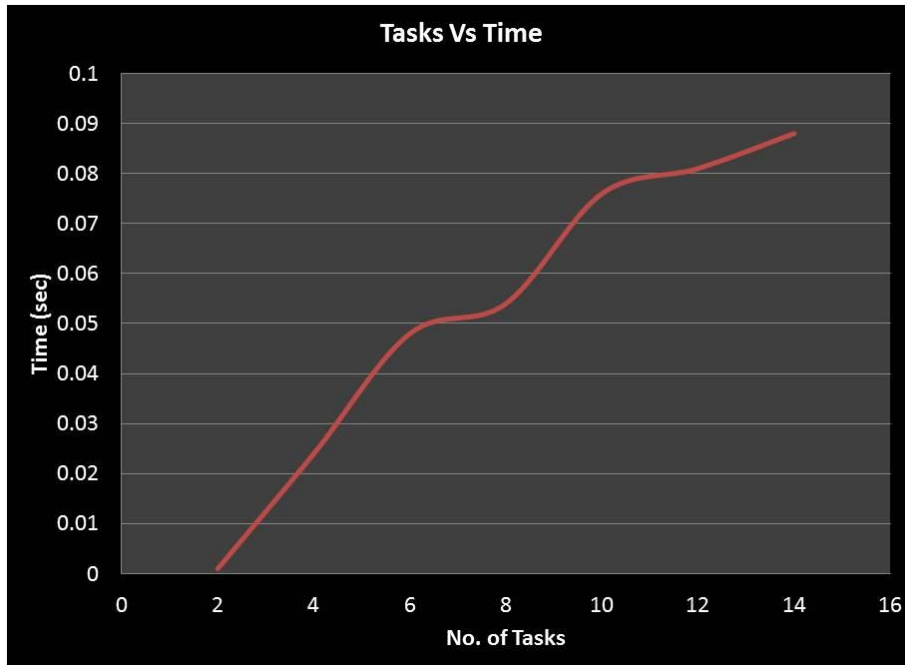


Figure 5.1: Tasks vs Time Trend

5.1.1 Tasks vs Time

The adopted technique incorporates Java front end UI, integration of Context Sensors in order to impart ability to re-plan and adapt at run-time, seamless transition of Domain and Problem descriptions in PDDL to Java and their parsing to Java Based JSHOP2 for extraction of requisite plan using Java back end. The complete cycle as depicted in Fig 3.1 when traversed should take considerably more time than existing non-adapting frameworks, but keeping in mind the performance aspect of approach the Goal Model is disintegrated into Sub-Goal Models; which are implemented with each having a considerably smaller domain, thus reducing the search space for each sub-problem and substantially enhancing its performance.

Re-planning required the multiple iterations of search space for reaching the most suited plan, this paradigm has been addressed by considering maximum possible scenarios (discussed below) that may pose user with unpredictable situations and incorporating them in Goal Model and generating Search Tree bifurcations at the First Step Depth. Fig 5.1 depicts the time consumption for tasks to fulfill the final goal state.

5.1.2 Evaluation Scenarios

Our CARE Reasoning application i.e. i*INSTA PLANNER is divided into two main modules. The first module covers the working day tasks (plan) of any professional e.g. surgeon, lecturer or business man etc. Second module is concerned with the weekend/holiday activities of individual. Following scenarios are built to verify the above mentioned architecture.

5.1.2.1 Scenario 1

This scenario is taken from the working day module of application which does not involve CARE technology.

- a. In this simple case planner application just remind user to carry different things (based on some initial conditions) before leaving for job and also suggest him some tasks that he has to perform before leaving home.
- b. This scenario does not involve input from the user and CARE context sensing mechanism.
- c. The generated plan is only dependent on the location of different thing that he has to pack before leaving home.
- d. Fig 5.2 show the plan suggested by daily planner of application as reminder of packing different things before leaving based on initial conditions and final goal state (without using CARE technology).

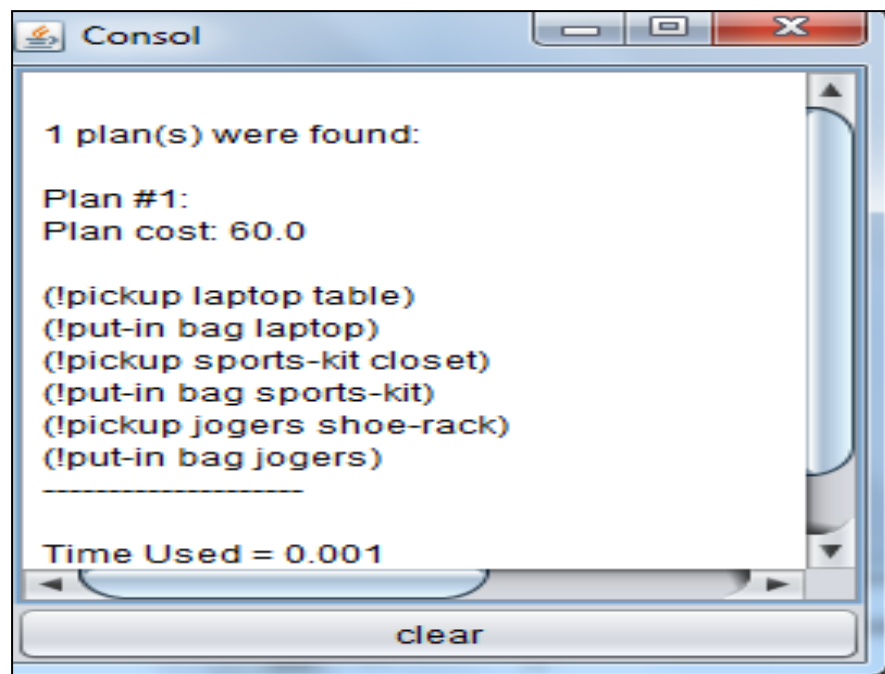


Figure 5.2: Plan

5.1.2.2 Scenario 2

This scenario is also taken from working day module but it involves CARE technology. In this case transport is suggested to the user and by using CARE methodology system itself analyzes the user preferences (based on some initial conditions), environmental conditions and user/system context.

Case1:

- a. John asks for application suggestion for his suitable mode of transportation to reach job place.
- b. The context sensing mechanism of application starts checking the weather forecast in his town.
- c. If weather is sensed pleasant and no forecast of rain is found then train is suggested as the suitable mode of transport for John.
- d. After suggesting train application checks if John has cash. If no cash is found then planner is again invoked for some new sequence of tasks (re-planning).
- e. Planner suggests John to walk to bank and draw cash, meanwhile application also check that whether the nearby ATM is working or faulty.
- f. Fig 5.3 shows the plan generated for John for his course of actions.

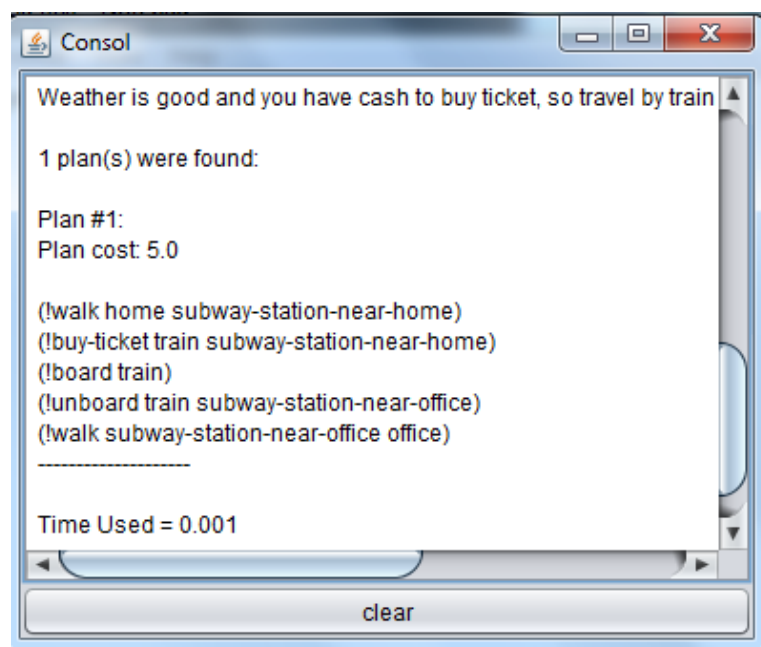
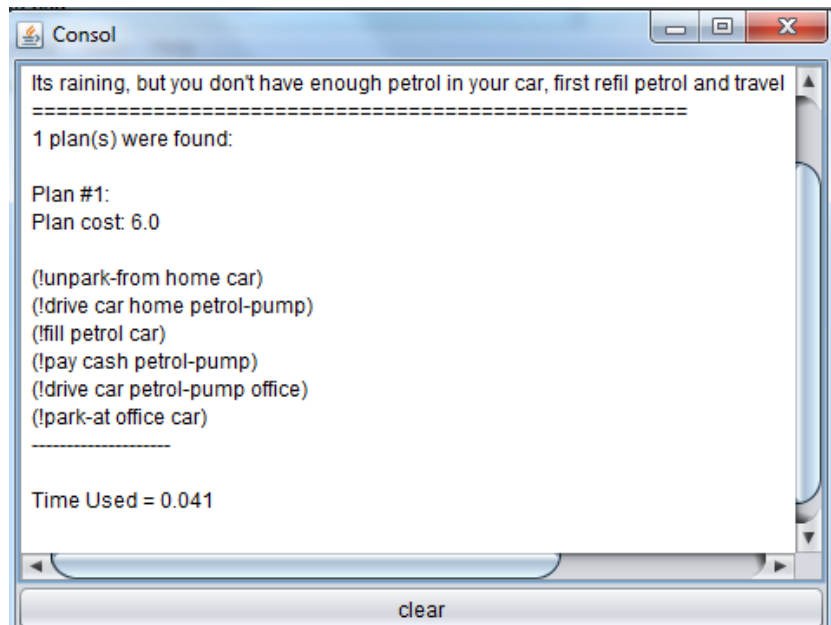


Figure 5.3: Plan

Case2:

- a. Any changes in weather conditions again activate the context sensing mechanism of application and it starts checking the weather conditions.
- b. If weather is sensed cloudy and forecast of rain is found then planner is again invoked and car is suggested as the suitable mode of transport for John.
- c. After suggesting car, application automatically checks if John has gas/petrol in car by fuel sensor. If no/less fuel is found, planner is again invoked which start re-planning according to the changed situation.
- d. Fig 5.4 shows the plan generated for John for his course of actions.



```
Consol
Its raining, but you don't have enough petrol in your car, first refill petrol and travel
=====
1 plan(s) were found:

Plan #1:
Plan cost: 6.0

(!unpark-from home car)
(!drive car home petrol-pump)
(!fill petrol car)
(!pay cash petrol-pump)
(!drive car petrol-pump office)
(!park-at office car)
-----

Time Used = 0.041

clear
```

Figure 5.4: Plan

5.1.2.3 Scenario 3

This scenario is taken from holiday module of i*INSTA PLANNER application in which system not only plan and re-plan for user but also perform some actions to facilitate him in achieving his goals(as suggested by planner).

- a. John wake up early morning and start i*INSTA PLANNER that play a role of his smart secretary.
- b. i*INSTA PLANNER automatically checks the time of day. Early morning it suggest John different actions that he has to perform in morning like Prepare Breakfast, Clean snow, Clean house etc.
- c. In afternoon planner is invoked automatically and suggest john that he has to perform some important tasks in afternoon like Do Laundry, Prepare Lunch etc. It also gives him some options for lunch based on his preference that whether he wants some Healthy Food or Instant Food. Moreover planner also suggests sequence of task to prepare selected lunch item.
- d. As soon as evening time is sensed i*INSTA PLANNER start generating different excursion plans for John like Visit Relatives, Go for Movie or Go for Dinner.
- e. System also perform some actions to help John to achieve final goal state for example if he selected Go for Dinner, application start searching for the nearby restaurants available in his city based on his preference of Chinese, Continental or Fast Food.
- f. If he selected Watch Movie as his goal then application give him options of nearest cinema in his town and current movies in that cinemas with their show timings. Application also gives him option for online reservation of seats to achieve hundred percent goal state.

Sr.no	Planning	Context Sensing	Adaptation	Re-planning
Scenario 1	<i>Yes</i>	<i>No</i>	<i>No</i>	<i>No</i>
Scenario 2	<i>Yes</i>	<i>Yes</i>	<i>Yes</i>	<i>Yes</i>
Scenario 3	<i>Yes</i>	<i>Yes</i>	<i>Yes</i>	<i>Yes</i>

Figure 5.5: Comparison of scenarios

Chapter 6

Conclusion & Future Work

6.1 Conclusion

In this thesis work we have proposed architecture to implement reasoning component of already proposed CARE framework which we named as Care Reasoning Framework (Figure 3.1). The proposed reasoning framework is based on control loop having three main components. It includes, Monitor/Evaluate component: which ensures that the information is received from the monitoring operations. The Planner component: responsible for activating the planner to generate sequence of tasks to determine adaptation. The Adaptation component: is mainly associated with the execution of the plan selected by the user.

This framework helps in dynamic problem formulating of the requirements of *Self-adaptive* systems. Dynamicity is due to the changes in user needs for example changes in user goals and preferences, contextual changes or changes in available resources.

So far we have implemented a prototype application to validate our architecture by integrating AI planner (JSHOP2) with our application which addresses user preferences at run-time and generate plans according to these preferences. Moreover application also continuously senses changes in its operational environment and re-plan according to these changes. System also performs some actions to facilitate user to achieve his goals which is actually the execution of generated plan.

6.2 Future work

We identified following future research direction during our work.

1. Enabling AI application to sense the changes in user intentions so that it can adapt and re-plan according to the changed mood and intentions of the user.
2. Deployment of large AI Planner based applications on Smart Phones.
3. Integration of planning application with service based applications.

BIBLIOGRAPHY

- [1] *Modeling Dimensions of Self-Adaptive Software Systems* Jesper Andersson¹, Rogerio de Lemos², Sam Malek³, Danny Weyns⁴.
- [2] *Self-Managed Systems: an Architectural Challenge* Jeff Kramer and Jeff Magee.
- [3] Proceedings of the 1st ACM SIGSOFT workshop on *Self-managed system* in D. Garlan, J. Kramer and A. Wolf, eds., ACM Press, Newport Beach, California, 2004, pp. 119.
- [4] Proceedings of the first workshop on *Self-healing systems*, in D. Garlan, J. Kramer and A. Wolf, eds., ACM Press, Charleston, South Carolina, 2002, pp. 120.
- [5] International Conference on *Self-Organization and Autonomous Systems in Computing and Communications (SOAS2006)*, Erfurt, Germany, September 2006
- [6] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimhigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum and A. L. Wolf, *An architecture-based approach to self-adaptive software, Intelligent Systems and Their Applications, IEEE [see also IEEE Intelligent Systems]*, 14 (1999), pp. 54-62.
- [7] D. Garlan and B. Schmerl, *Model-based adaptation for self-healing systems, Proceedings of the first workshop on Self-healing systems, ACM Press, Charleston, South Carolina, 2002.*
- [8] E. Gat, *Three-layer Architectures, Artificial Intelligence and Mobile Robots*, MIT/AAAI Press, 1997.
- [9] Sotirios Liaskos, Sheila A. McIlraith, Shirin Sohrabi, and John Mylopoulos. *Representing and reasoning about preferences in requirement engineering. Requirements Engineering*, 16:227-249, 2011. 10.1007/s00766-011-0129-9.
- [10] Qureshi, Nauman A., Liaskos, Sotirios, and Perini, Anna. *Reasoning About Adaptive Requirements for Self-Adaptive Systems at Runtime. In Proc. of the 2nd International Workshop on Requirements@Run.Time, pages 1622. IEEE, 2011b.*
- [11] Dardenne A, van Lamsweerde A, Fickas S (1993) *Goal-directed requirements acquisition. Sci Comput Program* 20(12):350
- [12] Yu ESK (1997) *Towards modeling and reasoning support for early-phase requirements engineering.* In: Proceedings of the 3rd IEEE international symposium on requirements engineering (RE97). Washington, DC
- [13] N. A. Qureshi, I. Jureta, and A. Perini, *Requirements engineering for self-adaptive systems: Core ontology and Problem statement, in 23rd Intl. Conf. on Advanced Information Systems Engineering (CAiSE11), ser. LNCS, vol. 6741. Springer, 2011, pp. 3347.*
- [14] B. H. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, *Software Engineering for Self-Adaptive Systems: A Research Roadmap, ser. LNCS. Springer, 2009, vol. 5525, pp. 126.*
- [15] N. A. Qureshi and A. Perini, *Engineering adaptive requirements, in ICSE Wks. on Software Engineering for Adaptive and Self-Managing Systems (SEAMS09), May 2009, pp. 126-131.*

- [16] Qureshi, N.A., Perini, A., *Requirements Engineering for Adaptive Service Based Applications, 2010 18th IEEE International Requirements Engineering Conference.*
- [17] Fickas, S. and Feather, M. *Requirements monitoring in dynamic environments, Proc. 2nd IEEE International Symposium on Requirements Engineering (RE'95), 1995.*
- [18] Salifu, M., Yu, Y., Nuseibeh, B. *Specifying Monitoring and Switching Problems in Context, Proc. 15th IEEE International Conference of Requirements Engineering (RE07), pp. 211-220, 2007.*
- [19] Berry, D., Cheng, B., and Zhang, J. *The four levels of requirements engineering for and in dynamic adaptive systems, Proc. 11th International Workshop on Requirements Engineering Foundation for Software Quality (REFSQ05), 2005.*
- [20] Loris Penserini, Anna Perini, Angelo Susi, and John Mylopoulos. *High variability design for software agents: Extending Tropos. TAAS, 2(4), 2007.*
- [21] Sotirios Liaskos, Alexei Lapouchnian, Yiqiao Wang, Yijun Yu, and Steve M. Easterbrook. *Configuring common personal software: a requirements driven approach. In 13th IEEE International Conference on Requirements Engineering, (RE05), Paris, France, pages 918, 2005.*
- [22] Qin Zhu, Lin Lei, Holger M. Kienle, and Hausi A. Muller. *Characterizing maintainability concerns in autonomic element design. IEEE International Conference on Software Maintenance (ICSM 2008), pages 197206, 28-2008-Oct. 4 2008.*
- [23] I. J. Jureta, J. Mylopoulos, and S. Faulkner. *Revisiting the core ontology and problem in requirements engineering. In 16th IEEE Int. Requirements Eng. Conf., pages 7180, 2008.*
- [24] M. S. Feather, S. Fickas, A. Van Lamsweerde, and C. Ponsard. *Reconciling system requirements and runtime behavior. In IWSSD 98: Proceedings of the 9th international workshop on Software specification and design, page 50, Washington, DC, USA, 1998. IEEE Computer Society.*
- [25] William Robinson. *A Roadmap for Comprehensive Requirements Monitoring. Computer, 43(5):6472, 2009.*
- [26] M. Salifu, Yijun Yu, and B. Nuseibeh. *Specifying monitoring and switching problems in context. 15th IEEE International Requirements Engineering Conference (RE 07), pages 211 220, Oct. 2007.*
- [27] M. S. Feather, S. Fickas, A. Van Lamsweerde, and C. Ponsard. *Reconciling system requirements and runtime behavior. In IWSSD 98: Proceedings of the 9th international workshop on Software specification and design, page 50, Washington, DC, USA, 1998. IEEE Computer Society.*
- [28] Betty H. C. Cheng, Holger Giese, Paola Inverardi, Jeff Magee, and Rogerio de Lemos. *Software engineering for self-adaptive systems: A research roadmap. In Software Engineering for Self-Adaptive Systems, volume 08031 of Dagstuhl Seminar Proceedings. Schloss Dagstuhl, Germany, 2008.*
- [29] Jon Whittle, Pete Sawyer, Nelly Bencomo, Betty Cheng, and Jean-Michel Bruel. *Relax: a language to address uncertainty in self-adaptive systems requirement. Requirements Engineering, 15:177196, 2010.*

- [30] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. *Goal-directed requirements acquisition*. *Sci. Comput. Program.*, 20(1-2):350, 1993.
- [31] Van Lamsweerde A (2001) *Goal-oriented requirements engineering: a guided tour*. In: *Proceedings of the fifth IEEE international symposium on requirements engineering, RE 01*. IEEE Computer Society, Washington, DC.
- [32] <https://code.google.com/p/planning4j/>
- [33] <https://github.com/gerryai/PDDL4j>
- [34] <http://www.antlr.org/>
- [35] Ilghami, O. (2005). *Documentation for JSHOP2*. Technical report CS-TR-4694, Department of Computer Science, University of Maryland.
- [36] D. Nau, T.-C. Au, O. Ilghami, U. Kuter, W. Murdock, D. Wu, and F. Yaman 2003 *SHOP2: An HTN Planning System*. To appear, *Journal Artificial Intelligence Research*.
- [37] Erol, K., Nau, D., Hendler, J. (1994). *HTN planning: Complexity and expressivity*. In *AAAI-94*.
- [38] <http://www.info.ucl.ac.be/research/projects/AVL/ReqEng.html>.
- [39] B. H. C. Cheng and J. Atlee, M., *Research Directions in Requirements Engineering*, in L. Briand and A. L. Wolf, eds., *Future of Software Engineering 2007*, IEEE-CS Press, 2007.
- [40] E. M. Dashofy, A. van der Hoek and R. N. Taylor, *Towards architecture-based self-healing systems*, *Proceedings of the first workshop on Self-healing systems*, ACM Press, Charleston, South Carolina, 2002.
- [41] N. Medvidovic, D. S. Rosenblum and R. N. Taylor, *A language and environment for architecture-based software development and evolution*, *Proceedings of the 21st international conference on Software engineering*, IEEE Computer Society Press, Los Angeles, California, United States, 1999.
- [42] H. Gomaa and M. Hussein, *Dynamic Software Reconfiguration in Software Product Families*, *5th International Workshop on Software Product-Family Engineering*, LNCS 3014, Springer 2004, 435-444., Siena, Italy, 2003.
- [43] S. Malek, et al. *A Framework for Ensuring and Improving Dependability in Highly Distributed Systems*. *Architecting Dependable Systems III*. Eds. R. de Lemos, C. Gacek, A. Romanovsky. LNCS 3549. Springer. Berlin, Germany. 2005.
- [44] S. Malek, C. Seo, S. Ravula, B. Petrus, and N. Medvidovic. *Reconceptualizing a Family of Heterogeneous Embedded Systems via Explicit Architectural Support*. *International Conference on Software Engineering (ICSE 2007)*. Minneapolis, Minnesota, May 2007.
- [45] R. Haesevoets, et al. *Managing Agent Interactions With Context-driven Dynamic Organizations*. *Engineering Environment-Mediated Multi-Agent Systems*. *Lecture Notes in Computer Science*, vol. 5049, 2007.
- [46] J. Andersson, et al. *An Adaptive High-Performance Service Architecture*. *ETAPS Workshop on Software Composition Electronic Notes Theoretical Computer Science 114*. 2005.

- [47] J. Kramer and J. Magee, *The evolving philosophers problem: dynamic change management*, *Software Engineering, IEEE Transactions on*, 16 (1990), pp. 1293-1306.
- [48] D. Hirsch, J. Kramer, J. Magee and S. Uchitel, *Modes for Software Architectures*, *Third European Workshop on Software Architecture (EWSA 2006)*, Springer, Nantes, France, Sept 2006.
- [49] McDermott, D.: *The 1998 AI planning systems competition*. *AI Magazine* 21 (2000)
- [50] Bacchus, F.: *The 2nd International Planning Competition home page*. <http://www.cs.toronto.edu/aips2000/> (2000)
- [51] Long, D., Fox, M.: *An overview and analysis of the results of the 3rd International Planning Competition*. *Journal of AI Research* 20 (2003)
- [52] Gerevini, A.: *The 5th international planning competition*. *ICAPS'06 Report (2006)*