Runtime Reasoning of Requirements for Self-Adaptive Systems using AI Planning Techniques

Zara Hassan[1]
Department of CS & IT
The University of Lahore
Lahore, Pakistan

Nauman Qureshi[2]
School of Electrical Engg. & Computer Science,
National University of Sciences and Technology,
Islamabad, Pakistan

Muhammad Adnan Hashmi[3], Arshad Ali[4]
Department of CS & IT
The University of Lahore
Lahore, Pakistan

Abstract—Over the years, the domain of Self-Adaptive Systems (SAS) has gained significant importance in software engineering community. Such SAS must ensure high customizability and at the same time effective reasoning to meet their objectives by meeting end-user goals more effectively and efficiently. In this context, techniques related to *Automated Planning* have acquired substantial precedence owing to their adaptability to diverse scenarios based upon their enhanced knowledge extraction from available Knowledge Base. These AI planning techniques help in supporting self-adaptation mechanism of SAS. We have investigated these techniques to perform runtime reasoning of SAS requirements. This paper proposes an architecture for implementing the reasoning component of previously proposed Continuous Adaptive Requirement Engineering (CARE) framework. The proposed architecture has been experimentally verified by implementation of a prototype application using JSHOP2 (Java implementation of SHOP2, an HTN Planner).

Keywords—Self-Adaptive Systems (SAS); reasoning; requirement engineering; AI planning; CARE framework; runtime reasoning of requirements

I. Introduction

The software systems are increasingly expected to satisfy their functional and non-functional requirements, even under changing conditions in their environments, including fluctuations in user demands (requirements), resource availability (system parameters) and the presence of cyber adversaries. Self-Adaptive Systems (SAS) address this need, since they are required to modify themselves according to the changes in the end user requirements or the environment in which they operate or the system parameters, to remain operational [2] [24]. Such systems have the ability to continuously monitor their own state and the state of their environment, and to autonomously change their structure and behavior to operate as best as possible, with respect to a defined goal in the presence of run-time changing conditions.

The desired end states (goals), as defined by the user and dictated by the system itself for computational purposes, are translated in the form of explicit requirements. Requirements engineering approach provides the basic considerations for determining the performance of evolved system. However, pragmatically the existing requirements engineering (RE) techniques work well where requirements of system are well understood at design time and evolve very slowly with respect to time. These techniques fail to provide solutions in abruptly changing requirements, hence rendering them unable to support Self-Adaptive Systems (SAS) where changes in all domains are very dynamic [1].

Self-Adaptive Systems manifest themselves into uncertainty in both context and lack of knowledge thanks to the ever-changing variables [25]. It leads to a scenario where system is made to take critical decisions for adapting itself with respect to set goals (whether to adapt or not, when to adapt, which adaptation technique to use, etc) in a dynamic and partially observable environment. The authors in [27] argue that contextual uncertainty in the operating environment requires to be reduced in order to improve the performance of SAS.

In the literature, a few approaches attempt to handle uncertainty merely by including it in the description; whereas some approaches rely on monitoring of context but lack the ability to alter the system. Hence, the environment remains uncontrollable and true implementation of adaptation logic [26] cannot be affected thus leading to undesired adaptation results. This paper proposes a model that enables the SAS to continuously monitor the contextual variations and is equipped with the mechanism to alter itself at runtime according to the altering requirements.

The environment modelling approach that we have adopted bears a few resemblances with the model of artifacts proposed in [28] and [29]. Major dissimilarity between these two approaches is that in [29] an artifact denotes a physical or computational entity in the environment (e.g. a mouse, a sensor, a web-service etc) whereas in [28] an artifact connotes a conceptual or logical entity (e.g. a car, a house, a place etc). However, our approach is hybrid where an artifact can be physical or logical entity.

In order to effectively reason about the modelled artifacts stemming from dynamic environment, this paper expands the Continuously Adaptive Requirement Engineering (CARE) framework proposed in [3]. CARE is both goal and user oriented RE framework. This paper extends the *reasoning component* of CARE framework, which provides effective decision making to meet the end-user *preferences* based on hatched *goal models*, by effective use of *AI planning* techniques.

Rest of the paper is organized as follows. Section II presents a brief ***literature survey*** leading to the research work. It is followed by ***proposed architecture*** of reasoning component in section III. A ***case study*** is presented in section IV that demonstrates the proposed architecture along with its goal model, the planning domain description representation and its integration with Java based Self-adaptive application. A brief ***evaluation of proposed model*** through developed prototype application is presented in section V. Section VI ***concludes*** the paper.

II. Literature Survey

This section investigates various semi-plugged in voids involved in the realization of SASs with particular focus on their implementation at runtime and their ability to meet the requirements posed by system, context and user.

A. Requirement Engineering for SAS

In software development life cycle, requirement engineering is the ground activity upon which the working of whole system depends. Work of Fickas and Feather presented in [17] on requirements monitoring is a key contribution towards run-time requirements. Continuous requirements monitoring is necessary because of the deviation of system behavior at run-time from requirements model, which ultimately triggers the demand for system moderation. Such deviations need to be agreed with the changing conditions of environment so that the reasons can be identified and suitable adaptation is achieved. This is called monitoring and switching by Salifu in [32].

Berry, Cheng, and Zhang identified four-level model for engineering requirements posed by dynamic adaptive systems [9]. Level 1 includes traditional RE activities done by analyst. Level 2 includes run-time adaptive requirements. Level 3 includes requirements engineering done by analyst to determine the adaptation mechanism which actually enables the system to adapt. Level 4 includes adaptation requirements that are associated to specific adaptation solutions.

We also critically analyzed applicability of existing goal-oriented RE approaches related to our work.

TROPOS methodology for goal modeling is used by Penserini et al. [4] 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 changes in user needs.

Liaskos et al. [31] uses requirements driven approach to address the problem of changing requirements by configuring software using goal-oriented approach. They model user's high level preferences as goal alternatives and then match them with the system's configurations. In this way, they support 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 et al. [30] uses 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 requirements engineering, Jureta [5] redefines the concepts of core requirements ontology. The core ontology is mainly based on goal oriented concepts and also on mentalistic notions which are called modalities.

KAOS [8] approach for requirements modeling focus on relating the functional and non-functional requirements to the enterprise goals because it assumes sufficient knowledge about the current organizational state. i* modeling approach [12] 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 affect the behaviors of actors.

The existing RE approaches discussed above 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 [3, 6] proposes a novel framework that captures and analyzes user requirements at run-time. This framework is called Continuous Adaptive Requirement Engineering framework.

B. Uncertainty in SASs

The systems having the capability to adapt and alter the involved players or scenario inherently require continuous interaction with the environment either to sense or to change. The continuous altering nature of environment and system thus places additional burden on system, of dealing with the introduced uncertainty as argued in [23].

An elaborate description of SASs, and evaluation of various approaches and models leading to declaration of CARE as the most effective framework for reasoning of requirements at runtime in presence of uncertainty in [22] gives a venue for further capitalization on the concept.

C. Goal Modeling and HTNs

In the field of requirements engineering (RE), goal oriented modeling approaches have acquired substantial attention as they enable the system to traverse the unexplained gap between stakeholder requirements (goals) and the instruments (actions/tasks/plans) whose manipulation ensure attainment of these goals. The presently employed goal-oriented modeling frameworks [8, 12] consider goals as mandatory requirements that must be satisfied by any suitable solution. However, these frameworks are unable to satisfy the preference requirements presented by stakeholders. So a framework [10] is introduced allowing users to specify both the preference requirements and priorities, which are later utilized to select the specifications that meet the mandatory requirements while best satisfying the preference requirements as per accorded priorities.

In HTN (Hierarchical Task Network) [18] there is a provision to manage mandatory goals alongside preference goals based on evaluation of quantized criterion. This is achieved by the arrangement of tasks in a hierarchical order and their recursive mitigation into other sub tasks.

HTN domain is composed of operators and methods that outline feasible maneuvers to achieve goals whereas HTN problem specification comprises a list of predicates and higher level tasks that are required to be completed for the attainment of goal state. HTN based planner initially scans the domain and specification of problem and then recursively performs HTN tasks to attain high level goal state. Hence, the mandatory decomposition is translated into a set of HTN operators, methods, and tasks, while the set of priorities and preference goals are converted into PDDL 3.0 preference constraints and metrics, modelled into weighted evaluation function for reasoning [21].

D. AI Planners

Planning problems are represented in PDDL, STRIPS or HTNs which are processed by AI planners to generate solutions i.e. task sequences [10]. In our self-adaptive application we are using JSHOP2 planner [11] which is Java implementation of SHOP2 [16]. The core functionality of JSHOP2 is constructed on planning formalism called hierarchical task network planning [15, 11].

In most of the automated planning systems, planners have to strike out various possibilities prior to discovering a workable plan/solution; as they perform a trial-and error search of a large solution space. On the contrary, HTN planners conduct this same very search by firstly applying HTN methods to decompose tasks into subtasks thus creating a planning problem network [14] called hierarchical network of tasks, which in turn can be efficiently searched by planner to generate the requisite task sequence.

E. CARE Framework

CARE [7] is goal and user oriented requirement engineering framework that captures and analyzes user requirements at runtime. The main idea behind CARE is that the system itself plays the role of analyst i.e. it performs RE activities at runtime to satisfy end user preferences and adapt itself to meet changes in user goals and preferences. To achieve this adaptation, system automatically updates its knowledge about the operational environment and end user needs. The requirements captured by system at runtime are called service requests [5] in CARE which can be expressed in XML format. These service requests either consist of 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 requirements data. First it evaluates the incoming data to determine which type of adaption is required [6], before activating a planner. After evaluating, the **Plan** activity activates the planner to generate task sequences and the selected plan is executed by Reasoner's **Adapt** activity [7].

III. Proposed Architecture

Fig.1 shows the revised architecture of CARE reasoning framework integrating goal modeling and automated planning techniques [12, 13]. The requirements are represented as goal model [10] that not only represents the functional requirements but also the user's 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 [7] by using HTN semantics. This action theory represents 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/agents automatically interact with each other to support runtime reasoning of requirements. It consists of following four agents:

- User Agent
- Planning Agent
- Lookup Agent
- Update Agent

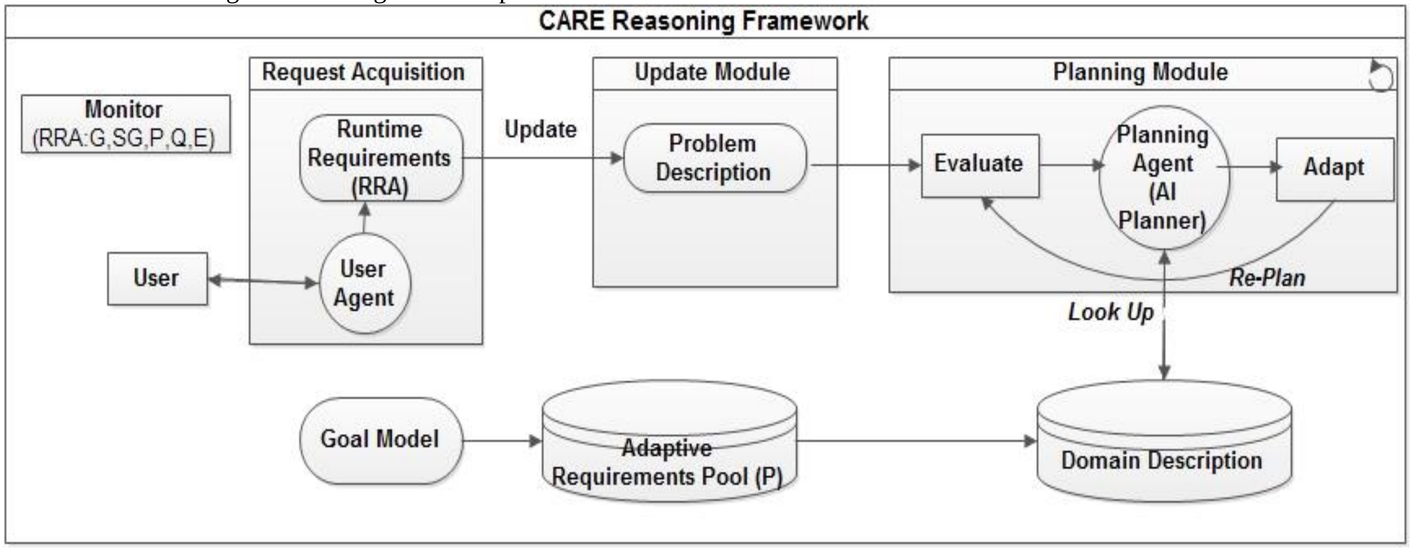


Fig. 1. Revised CARE Reasoning Framework.

The requirements which are called runtime requirements artifacts (RRA) are captured from the user through the ***user agent***. These RRAs consist of various requirement elements e.g. user hard and soft goals (G, SG), preferences (P), quality criteria (Q) and the data received from monitors which sense the changes in environmental conditions (E). These user RRAs are transformed into a complete problem description where the values of monitored variables formulate the initial conditions, user goals formulate the goal specification, and quantitative prioritization of preference goals covers the preferences specification [7]. The problem description is then input to the ***planning agent*** which first evaluates the information received from monitors and determines which type of adaptation is required. It then activates 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 tasks) in domain description that satisfies user goals defined in the given problem description. The resulting plan is displayed to the user again through the ***user agent***. Re-planning is required if some change is sensed in the environmental conditions or the prescribed plan is not executed as desired, thus warranting generation of a new plan having different initial conditions. The ***update agent*** is responsible for updating the problem description according to the new requirements.

IV. Case Study

We demonstrate the reasoning architecture presented in Fig.1 with the help of a prototype application aimed at acting as a Virtual Secretary to the user. The application covers basic daily life tasks of various professionals e.g. Lecturer, Surgeon and Businessman with the flexibility 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 Fig.2). Figure demonstrates the identification of set of goals, set of AND/OR decomposed tasks along with their pre-requisite conditions to meet the goals, predefined methods encapsulating various tasks in order for meeting minor goals, system ground state conditions and in addition a pool of adaptive requirements.

The tasks are categorized 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. Moreover, the application continuously senses the changes in its operational environment and re-plans according to these changes. For example, the system notifies ***Faulty ATM Machine*** and suggests user to ***Use Cheque*** to ***Draw Cash***. Preference goals are also supported 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.

A. HTN based Goal Model

The prototype application based on the above mentioned scenario in order to extend the desired features of adaptability and handling of runtime requirements entails a Goal Model that encompassed a few possible eventualities and recursive corrective actions. A segment of the Goal Model is presented in Fig.2. The given model is of transportation module which depicts the possibility of reaching the work place via three different modes i.e. ***By Walk, By Subway*** and ***By Car***. Selection of a mode is done upon weighing the resultant of selected user preference and environmental variables like ***IsRaining***, ***IsUrgent*** and ***HaveCash***. Each mode of transport is further disintegrated into sub tasks through AND/OR decomposition, rendering it an HTN Model.

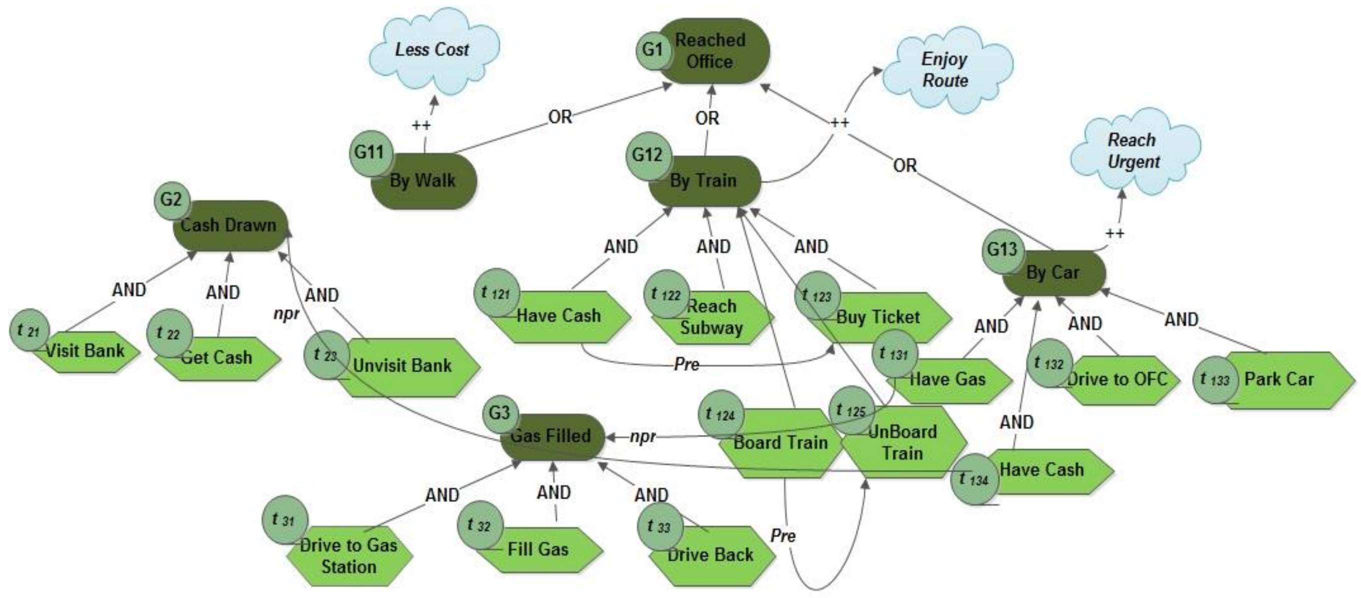


Fig. 2. Goal Model.

A plan [10] is devised for root goal satisfaction. Plan is a sequence of leaf level tasks that satisfy the root goal and is acquired by employing various sorts of tree searches e.g. A* Search, Depth First and Breadth First etc. 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_{21} ,t_{22} ,t_{23} ,t_{31} ,t_{32} ,t_{33} ,t_{132} ,t_{133}. Goal model in Fig.2 is a subset of i* Strategic Rationale Diagram [12, 19] but PRECEDENCE LINKS are additional to the core concept of i*. This concept of precedence links is used for goal modeling in [10, 20]

B. *Mapping Goal Model to PDDL using HTN Semantics*

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 richer representation of domain and its states.

1) Planning domain: Domain description is the knowledge base we prepare for the planner in order to enable it to solve the problems. 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. 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. A JSHOP2 compatible translation of prior mentioned scenario through its equivalent logical model is accomplished keeping in view the possible requirements that might be brought forth at runtime.

For the Transport Module of the application, the modes of transport are 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 is represented by means of methods, which are further subdivided into a set of ordered primitive tasks. Since the selection of mode of transportation is to be made upon evaluation of user based preferences, axiom of (***:- Mode-Sel ?Mode***) is used to reason/evaluate and choose the preferred method of reaching the destination work place (Listing 1).

Listing 1: Planning Domain

<u>Domain Axioms</u>:
Is-Urgent(X) =>Mode-Sel(X)

Is-Enjoyable(X) => Mode-Sel(X)

Cost-eff(X) =>Mode-Sel(X)

<u>Methods & Operators</u>:

Operator: *Walk (Loc-from,Loc-to)*

Pre = *Is-At (Loc-from)*

Eff = *Is-At (Loc-to)*

Operator: *Drive (Mode-car,Loc-from,Loc-to)*

Pre = *Mode-Sel(Mode-car)∧ Car-At(Loc-from)∧ Have-Gas(Mode-Sel)*

Eff = *Car-At(Loc-to) ∧Is-At(Loc-to)*

Operator: *Ride (Mode-train, Loc-from, Loc-to)*

Pre **=** *Mode-Sel(Mode-train) ∧ Is-At(Loc-from) ∧ Have-ticket(Mode-train)*

Eff = *Is-At(Loc-to)*

Method: *Via-Car(Mode-Car, Loc-from, Loc-to)*

Pre **=** *Car-At(Loc-to) ∧ Is-At(Loc-from) ∧ Need-Gas(Mode-Car)*

Tasks **=** *{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)}*

2) Planning problem: Planning problem description is the precise description of planning problem at hand i.e. the initial or current state and the goal state or the tasks to be realized. Problem description also includes the identification of various object types i.e. actors in the planning problem along with problem specific knowledge.

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 domain includes the declaration of Objects i.e. *Transport (walk, car, and train), Locations (Home, Gas-Station, Bank, Work-Place, Subway-Station-A, Subway-Station-B)* and their user defined metric preferences for each. The system ground state is defined by manipulating the predicate variables already defined in corresponding Domain Description. The problem satisfier Goal i.e. *Reaching-Workplace* is expressed containing evaluating variables (Listing 2).

Listing 2: Planning Problem

Initial State:

{Is-Urgent(Car), Is-Enjoyable(Train), Is-Cost-eff(walk),

Need-Gas (Car), Have-Cash (Cash),….}

Goals:

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

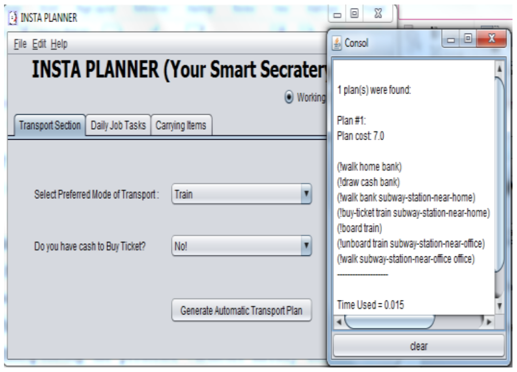
C. Development of CARE Reasoning Framework

Prototype application named INSTA PLANNER is developed to validate the 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. INSTA PLANNER is AI-Planner based self-adaptive application that incorporates online Web Services and 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 Listing 3, implementation of transportation module of application is explained with the help of an algorithm.

Listing 3: Algorithm for Transportation Module

```
GeneratePlanForTransportation()
    PlanGenerationMode ← GetPlanGenerationModeFromUser();
    if  PlanGenerationMode = AUTO
        UserRRA ← RRAsFromConextSensors();
    else
        UserRRA ← UserPrefrencesForTransportation();
    ProblemSpecifications ← GenerateProblemSpecification (UserRRA);
    Invoke PlanningAgent (ProblemSpecification);
    Plan ← SearchPlan(DomainDescription);
    return Plan;
UserPreferncesForTransportation()
    ModeOfTransportation ← GetModeOfTransportation();
    AdditionalInfo ← GetAdditionalInfo(ModeOfTransportation);
    UserRRA ← GetUserRRA ( ModeOfTransportation, AdditonalInfo);
    return UserRRA;
RRAsFromConextSensors()
    WeatherCondition ← WeatherWebservice();
    FuelLevel ← FuelMointering(FuelContextSensor)
    CashStatus ← CashMointering(CashContextSensor)
    AtmWorkingStatus ← PullInfoFromBank(ATMService);
    UserRRA ← GetUserRRA(WeatherCondition, FuelLevel,
                         CashStatus, AtmWorkingStatus);
    return UserRRA;
```



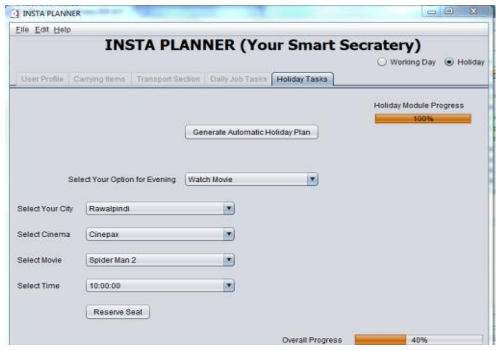
Select Preferred Mode of Transport : Train

Do you have cash to Buy Ticket? No!

Generate Automatic Transport Plan

```
1 plan(s) were found:

Plan #1:
Plan cost: 7.0

(!walk home bank)
(!draw cash bank)
(!walk bank subway-station-near-home)
(!buy-ticket train subway-station-near-home)
(!board train)
(!unboard train subway-station-near-office)
(!walk subway-station-near-office office)
--------------------

Time Used = 0.015
```

Fig. 3. INSTA Planner.



Working Day [✓ Holiday]

Holiday Module Progress 100%

Select Your Option for Evening: Watch Movie

Select Your City: Rawalpindi

Select Cinema: Cinepax

Select Movie: Spider Man 2

Select Time: 10:00:00

Overall Progress 40%

Fig. 4. INSTA Planner.

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. The screen shots presented in Fig.3 & Fig. 4 show the implementation of application with respect to above goal models.

V. Evaluation

The adopted technique incorporates Java front end UI integration of context sensors in order to impart ability to re-plan and adapt at runtime, seamless transition of domain and problem descriptions in PDDL to Java, and their parsing to Java based JSHOP2 for extraction of requisite plan. The complete cycle as depicted in Fig.1 when traversed should take considerably more time than existing non-adapting frameworks, but keeping in mind the performance aspect of proposed approach, the goal model is dis-integrated 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 requires the multiple iterations of search space for reaching the most suitable plan. This is addressed by considering maximum possible scenarios (discussed below) that may pose user with unpredictable situations and incorporate them in goal model and generate search tree bifurcations. Fig.5 depicts the time consumption for tasks to achieve the final goal state shown in the goal model (Fig.2).

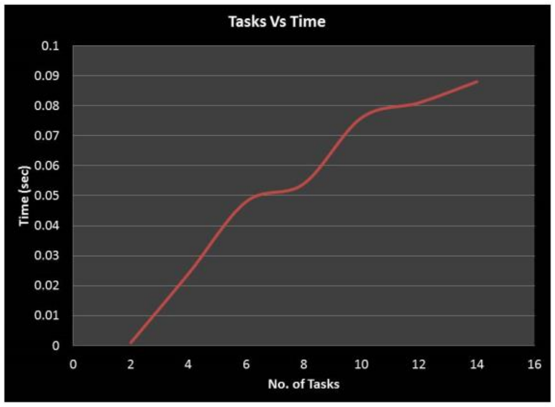
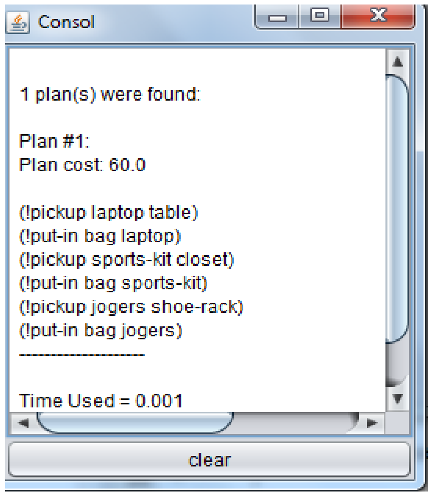


Fig. 5. Evaluation Trend.



```
1 plan(s) were found:

Plan #1:
Plan cost: 60.0

(!pickup laptop table)
(!put-in bag laptop)
(!pickup sports-kit closet)
(!put-in bag sports-kit)
(!pickup jogers shoe-rack)
(!put-in bag jogers)
--------------------

Time Used = 0.001
```

Fig. 6. Plan 1 generated by INSTA PLANNER without using CARE.

INSTA PLANNER is divided into two main modules. The first module covers the working day tasks of any professional. Second module is concerned with the weekend/holiday activities of individual. Following scenarios are developed to verify the proposed architecture.

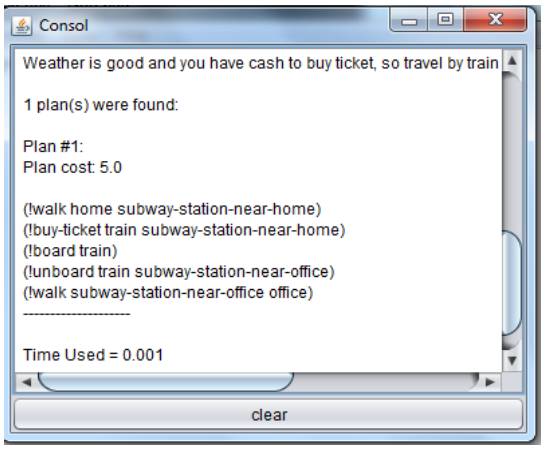
Scenario 1: This scenario is taken from the working day module of application which does not involve CARE technology.

- In this simple case, planner application just reminds user to carry different things (based on some initial conditions) before leaving for job and also suggests some tasks that he has to perform before leaving home.
- This scenario does not involve input from the user and CARE context sensing mechanism.
- The generated plan is only dependent on the location of different things that he has to pack before leaving home.
- Fig. 6 shows the plan suggested by daily planner of application as a reminder of packing different things before leaving based on initial conditions and final goal state (without using CARE technology).

Scenario 2: This scenario is also taken from working day module but it involves CARE technology. In this case transportation is suggested to the user and by using the CARE methodology, system itself analyzes the user preferences (based on some initial conditions, environmental conditions and user/system context).

Case 1:

- John asks for application suggestion for his suitable mode of transportation to reach job place.
- The context sensing mechanism of application starts checking the weather forecast in his town through web service agent.
- If the weather is sensed as pleasant and no forecast of rain is found then train is suggested as the suitable mode of transportation for John.
- 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).
- Planner suggests John to walk to bank and draw cash, meanwhile application also checks whether the nearby ATM is functional or faulty.
- Plan generated for John is shown in Fig. 7.

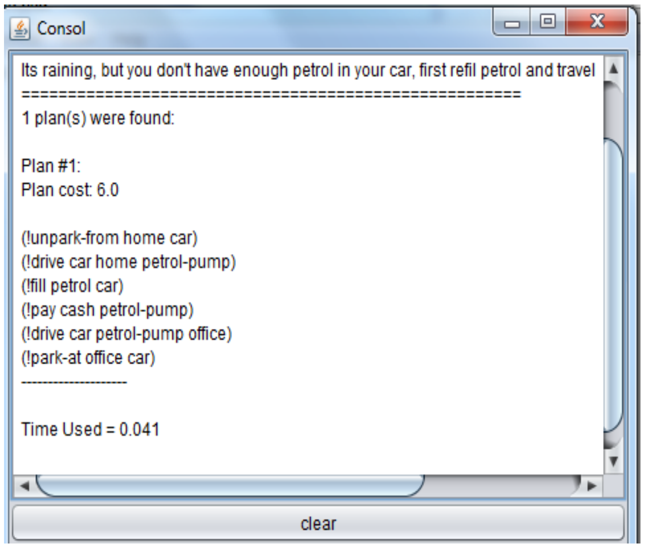


```
Weather is good and you have cash to buy ticket, so travel by train

1 plan(s) were found:

Plan #1:
Plan cost: 5.0

(!walk home subway-station-near-home)
(!buy-ticket train subway-station-near-home)
(!board train)
(!unboard train subway-station-near-office)
(!walk subway-station-near-office office)
--------------------

Time Used = 0.001
```

Fig.7. Plan 2 generated by INSTA PLANNER by using CARE.

Case 2:

- Any changes in weather conditions again activate the context sensing mechanism of application and it starts checking the weather conditions.
- 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.
- 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 starts re-planning according to the changed situation.
- Finally, the plan shown in Fig. 8 is generated for John for his course of actions.



```
Its raining, but you don't have enough petrol in your car, first refil 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
```

Fig.8. Plan 3 generated by INSTA PLANNER by using CARE.

Scenario 3: This scenario is taken from holiday module of INSTA PLANNER application in which system not only plans and re-plans for user but also performs some actions to facilitate him in achieving his goals (as suggested by planner).

- John wakes up early morning and starts INSTA PLANNER that plays the role of his smart secretary.
- INSTA PLANNER automatically checks the time of day. As it is early in the morning, it suggests different actions to John that he has to perform in morning like Prepare Breakfast, Clean snow, Clean house etc.
- In the afternoon, planner is invoked automatically and suggests to 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 tasks to prepare selected lunch item.
- As soon as evening time is sensed INSTA PLANNER starts generating different excursion plans for John like Visit Relatives, Go for Movie or Go for Dinner.
- System also performs some actions to help John to achieve his final goal state, for example if he selects Go for Dinner, application starts searching for the nearby restaurants available in his city based on his preference of Chinese, Continental or Fast Food and also helps in reservation of table in his selected restaurant.
- If he selected Watch Movie as his goal then application gives him options of nearest cinema in his town and current movies in that cinema with their show timings. Application also gives him the option for online reservation of seats. Hundred percent goal state is achieved when application gives confirmation of seats reservation via email.

Table 1 presents a comparison of these three scenarios.

TABLE I. COMPARISON OF SCENARIOS

Sr. No	Planning	Context Sensing	Adaptation	Re-planning
Scenario 1	*Yes*	*No*	*No*	*No*
Scenario 2	*Yes*	*Yes*	*Yes*	*Yes*
Scenario 3	*Yes*	*Yes*	*Yes*	*Yes*

VI. CONCLUSION AND FUTURE WORK

This paper presents an adaptive reasoning mechanism for the run-time requirements in Self-Adaptive Systems (SAS). We have implemented a prototype application to validate our architecture by integrating AI planner with our application which addresses user preferences at runtime and generates plans according to these preferences. Moreover the said application also continuously senses changes in its operational environment and re-plans according to these changes. System also performs some actions to facilitate user to achieve his goals which is actually the execution of the selected plan.

Currently we are working on enabling the 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.

REFERENCES

[1] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein. *"Requirements-aware systems. A research agenda for RE for self-adaptive systems"*. In Proceedings of 18th International Requirements Engineering Conference, pages 95-103. IEEE, 2010.

[2] J. Anderson, R. de Lemos, S. Malek, D. Weyns. *"Modeling dimensions of self-adaptive software systems"*. In Betty H. C. Cheng et al., editors, LNCS Hot Topics on Software Engineering for Self-Adaptive Systems. Springer, 2009.

[3] N. A. Qureshi and A. Perini, *"Requirements engineering for adaptive service based applications"* in 18th IEEE Intl. Requirements Engineering Conf., pp. 108–111, sept. 2010.

[4] L. Penserini, A. Perini, A. Susi, and J. Mylopoulos. *"High variability design for software agents: Extending Tropos"*. TAAS, 2(4), 2007.

[5] I. Jureta, S. Faulkner, and P. Thiran, *"Dynamic requirements specification for adaptable and open service systems"* in 15th IEEE Intl. Requirements Engineering Conf. pp. 381– 382, 2007.

[6] N. A. Qureshi, A. Perini, N. A. Ernst, and J. Mylopoulos, *"Towards a continuous requirements engineering framework for self-adaptive systems"* in First Intl. Workshop on Requirements@ Run.Time, pp. 9 – 16, sept. 2010.

[7] N. A. Qureshi, S. Liaskos, and A. Perini. *"Reasoning about adaptive requirements for self-adaptive systems at runtime"*. In Proceedings of 2nd International Workshop on Requirements@run.time, page 16–22, August. 2011.

[8] Dardenne A, van Lamsweerde A, Fickas S *"Goal-directed requirements acquisition"*. Sci Comput Program 20(1–2):3–50. 1993.

[9] 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 (REFSQ'05), 2005.

[10] S. Liaskos, S. A. McIlraith, S. Sohrabi, and J. Mylopoulos. *"Representing and reasoning about preferences in requirements engineering"*. Requirements Engineering, 16:227–249, 2011. 10.1007/s00766-011-0129-9.

[11] Ilghami, O. *"Documentation for JSHOP2"*. Technical report CS-TR-4694, Department of Computer Science, University of Maryland. 2005.

[12] Yu ESK "*Towards modelling and reasoning support for early-phase requirements engineering"*. In: Proceedings of the 3rd IEEE international symposium on requirements engineering (RE'97). Washington, DC. 1997.

[13] Van Lamsweerde A *"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. 2001.

[14] Erol, K., Nau, D., & Hendler, J. *"HTN planning: Complexity and expressivity"*. In AAAI-94. 1994.

[15] Nau, D., T. C. Au, O. Ilghami, U. Kuter, D. Wu, F. Yaman, H. Munoz - Avila, and J. W. Murdock. *"Applications of SHOP and SHOP2"*. Intelligent Systems, 20(2):34–41. 2005.

[16] D. Nau, T.-C. Au, O. Ilghami, U. Kuter, W.Murdock, D. Wu, and F. Yaman *"SHOP2: An HTN Planning System"*. Journal of Artificial Intelligence Research. (2003).

[17] Fickas, S. and Feather, M. *"Requirements monitoring in dynamic environments"*, In Proc. of 2nd IEEE International Symposium on Requirements Engineering (RE'95), 1995.

[18] Nau D, Cao Y, Lotem A, noz Avila HM. *"SHOP: simple hierarchical ordered planner"*. In: Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99), pp 968–973. 1999.

[19] Wang X, Lesperance Y. *"Agent-oriented requirements engineering using ConGolog and i*"*. In: AOIS-2001 Bi-conference workshop at agents 2001 and CAiSE'01. 2001.

[20] Fuxman A, Liu L, Mylopoulos J, Pistore M, Roveri M, Traverso P *"Specifying and analyzing early requirements in Tropos"*. Requir Eng 9(2):132–150. 2004.

[21] Gerevini A, Long D *"Plan constraints and preferences in PDDL3. Technical report"*. Department of Electronics for Automation, University of Brescia. 2005.

[22] Krupitzer, Christian & Maximilian Roth, Felix & VanSyckel, Sebastian & Schiele, Gregor & Becker, Christian..*"A survey on engineering approaches for self-adaptive systems"*. Pervasive and Mobile Computing. 17. 10.1016/j.pmcj.2014.09.009. 2014.

[23] Zavala, Edith & Franch, Xavier & Marco, Jordi & Knauss, Alessia & Damian, Daniela. *"SACRE: Supporting contextual requirements' adaptation in modern self-adaptive systems in the presence of uncertainty at runtime"*. Expert Systems with Applications. 98. 10.1016/j.eswa.2018.01.009. 2018.

[24] Shang-Wen Cheng and David Garlan. *"Stitch: A language for architecture-based self-adaptation"*. Journal of Systems and Software 85, 12, 2860–2875. 2012.

[25] N. Esfahani and S. Malek. *"Uncertainty in self-adaptive software systems"*. In Software Engineering for Self-Adaptive Systems II, Rogério de Lemos, Holger Giese, Hausi A. Müller, and Mary Shaw (Eds.). Lecture Notes in Computer Science, Vol. 7475. Springer Berlin Heidelberg, 214–238. 2013.

[26] J. Andersson, L. Baresi, N. Bencomo, R. deLemos, A. Gorla, P. Inverardi, T. Vogel, *"Software engineering processes for self-adaptive systems"*, in: Software Engineering for Self- Adaptive Systems II , in : LNCS,vol.7475,pp.51–75, Springer,2013.

[27] Gabriel A. Moreno, Javier Cámara, David Garlan and Mark Klein.*"Uncertainty Reduction in Self Adaptive Systems"*. In Proc. of the 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), Gothenburg, Sweden, 28-29 May 2018.

[28] M. A. Hashmi, A. E. Seghrouchni and M. U. Akram, *"A Planning Based Agent Programming Language Supporting Environment Modeling,"* 2015 IEEE / WIC / ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT), Singapore, pp. 76-83.doi:10.1109/WI-IAT.2015.22.

[29] A. Ricci, M. Piunti, and M. Viroli, *"Environment programming in multi-agent systems: an artifact-based perspective"* Autonomous Agents and Multi-Agent Systems, vol. 23, no. 2, pp. 158–192, 2011.

[30] 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 197-206, 28-2008-Oct. 4 2008.

[31] S. Liaskos, A. Lapouchnian, Y. Wang, Y. 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.

[32] 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.