Enabling Proactivity in Context-aware Middleware Systems by means of a Planning Framework based on HTN Planning

Preeti Bhargava, Ashok Agrawala
Department of Computer Science, University of Maryland, College Park
prbharga,agrawala@cs.umd.edu

ABSTRACT

Today's context-aware systems tend to be reactive or 'pull' based - the user requests or queries for some information and the system responds with the requested information. However, none of the systems anticipate the user's intent and behavior, or take into account his current events and activities to pro-actively 'push' relevant information to the user. On the other hand, Proactive context-aware systems can predict and anticipate user intent and behavior, and act proactively on the users' behalf without explicit requests from them. Two fundamental capabilities of such systems are: prediction and autonomy. In this paper, we address the second capability required by a context-aware system to act proactively i.e. acting autonomously without an explicit user request. To address it, we present a new paradigm for enabling proactivity in context-aware middleware systems by means of a Planning Framework based on HTN planning. We present the design of a Planning Framework within the infrastructure of our intelligent context-aware middleware called Rover II. We also implement this framework and evaluate its utility with several use cases. We also highlight the benefits of using such a framework in dynamic ubiquitous systems.

Categories and Subject Descriptors

C.3.2 [**[Special-Purpose and Application-based Systems]**]: Real-time and embedded systems, Ubiquitous computing

General Terms

Design, Experimentation, Human Factors, Performance

Keywords

Proactive computing, Context-aware computing and systems, Mobile and Ubiquitous systems, HTN Planning

1. INTRODUCTION

Today's context-aware systems tend to be 'reactive' or pull based - the user requests or queries for some information and the system responds with the requested information. These systems provide personalized and relevant information by filtering the information retrieved based on the user's preferences or limited context such as time, location, or web history. Such systems, once queried, could return a list of restaurants ordered by food preferences and sometimes recent browsing history. However, none of the systems anticipate the user's intent and behavior, or take into account his current events and activities to act 'pro-actively' and push relevant information to the user.

The initial notion of proactive computing, as proposed by Tennenhouse [14] and Want et al. [17], focused on human -supervised operations where the user stays out of the loop as much as possible until he is required to provide guidance in critical decisions. Tennenhouse [14] also stated that a fundamental goal of proactive computing is to enable autonomy in ubiquitous systems. However, Want et al. [17] specified key differences between proactive and autonomic computing and outlined several principles underlying proactive computing. Some of these are anticipation, context-awareness and statistical reasoning. Salovaara and Oulasvirta [11] discussed the general concept of proactive computing and suggested that a system can act proactively if it can hypothesize what its user's goals are.

Thus, for a ubiquitous or context-aware system to be effectively proactive, it is crucial that it tracks and predicts user intent [12] in order to take actions on the users' behalf without explicit requests from them. We term these systems as *Proactive context-aware systems*. These systems continuously sense and anticipate users' behavior - they acquire data from multiple sources and sensors, and then analyze the data in order to learn and predict users' behavior. Once the user behavior has been predicted, the system can pro-actively take actions on behalf of the users without an explicit request. These actions can include sending an email on the user's behalf, calling a phone number, changing the device mode (say from silent to ringer) based on his situation, or even booking movie tickets for the user. From this discussion, two fundamental capabilities of proactive context-aware systems emerge: *prediction* and *autonomy*.

In our previous work [1], we have addressed the first aforementioned capability of a proactive context-aware system - modeling and predicting user behavior, as part of the Rover II context-aware middleware [2, 6]. In this paper, we address the second key capability required by a context-aware system to act proactively - acting autonomously without an explicit user request. To address it, we present a new paradigm for enabling proactivity in context-aware middleware systems by means of a Planning Framework based on HTN planning. This framework is based on a predicate model of ubiquitous computing where the state of the context-aware system is represented as a set of variable bindings. It receives information about a task or activity that needs to be performed on the user's

MOBIQUITOUS 2015, July 22-24, Coimbra, Portugal

DOI 10.4108/eai.22-7-2015.2260304

behalf (which the user may have requested explicitly or implicitly) and generates a plan to achieve it. It utilizes the current context of the user, and internal and external information sources available to the system in order to determine the sequence of actions that should be performed in order to achieve the task.

The use of AI planning enables the system to decide, dynamically, how best to achieve user goals. It relieves users from the burden of having to know exactly what actions or tasks can or cannot be performed by the system and how to perform those actions. More importantly, since the system plans the sequence of actions to achieve the user's goals dynamically, it can adapt more easily to changing context and availability of resources. This also allows the system to handle faults that may occur while achieving the user's goals gracefully and with minimum user intervention.

Thus, our contributions in this paper are:

- We propose the paradigm of enabling proactivity in context-aware middleware systems by means of HTN Planning.
- We present the design of a Planning Framework within the infrastructure of our intelligent context-aware middleware called Rover II.
- We implement this framework and evaluate its utility with several use cases.
- We also highlight the benefits of using such a framework in dynamic ubiquitous systems.

The rest of the paper is organized as follows: Section 2 describes a scenario to motivate the need of enabling proactivity in a context-aware system. Section 3 provides an overview of HTN Planning. Section 4 describes the design of our planning framework and algorithm while Section 5 describes the implementation details and use cases. We discuss related work in Section 7 and conclude and outline future work in Section 8.

2. MOTIVATING SCENARIOS

In order to motivate the proactivity paradigm, we describe a simple scenario.

> A user is running late for a meeting with a colleague. The context-aware system infers this delay from his current location and from his meeting schedule (date, time, location, agenda, participants) marked in his calendar. It pro-actively communicates a message to the colleague with whom the meeting is scheduled to take place informing him/her of the delay.

There may be multiple ways of performing this task and some ways may be better than others because of the user's context or availability of information sources. In addition, the best way of achieving the goal may also change with time because of dynamically changing context. Also, depending on the current state of the system, different actions may need to be taken to achieve it. Hence, it is not easy to statically specify how the task is to be performed. Thus, new techniques are required that can dynamically perform such tasks for users.

Our proposed planning framework helps address this complexity and dynamism of context-aware systems. The framework analyzes the different ways in which a task can be achieved based on the available sources and user's context. It then determines the most feasible way of performing the task and generates a sequence of actions required to achieve it. In the aforementioned scenario, there are two possible ways of communicating with the user's colleague:by SMS or by email. SMS might be a faster way to reach the colleague as he/she may be driving or walking and may not check email soon. However, if the system doesn't have access to the phone number of the colleague then email is the only solution to reach him/her. Hence, the planning framework has to take all these factors into consideration and generate a sequence of actions that will be executed in order to perform this task. From the scenario, it is evident that an AI Planning technique such as HTN Planning can be applied to facilitate proactivity.

3. HIERARCHICAL TASK NETWORK PLANNING

Hierarchical Task Network (HTN) planning is an Artificial Intelligence (AI) planning technique [5]. The objective of an HTN planner is to produce a sequence of actions that perform some activity or task. The description of a planning domain includes a set of operators or primitive tasks and also a set of methods, each of which is a prescription for how to decompose a compound or non-primitive task into subtasks (smaller primitive tasks). Given a planning domain, the description of a planning problem will contain an initial state and a partially ordered set of tasks to accomplish.

HTN Planning proceeds by using the methods to decompose tasks recursively into smaller and smaller subtasks, until the planner reaches primitive tasks that can be performed directly using the planning operators. Thus, for each non-primitive task, the planner chooses an applicable method, instantiates it to decompose the task into subtasks, and then chooses and instantiates methods to decompose the subtasks even further if required. If the plan later turns out to be infeasible, the planning system will need to backtrack and try other methods.

HTN Planning requires well-conceived and well-structured domain knowledge. Such knowledge contains rich information and guidance on how to solve a planning problem. This structured and rich knowledge gives a primary advantage to HTN planners in terms of speed and scalability when applied to real-world problems. Examples of HTN Planners include Nets Of Action Hierarchies (NOAH) [10], System for Interactive Planning and Execution (SIPE) [18], Universal Method Composition Planner (UMCP) [4] and Simple Hierarchical Ordered Planner (SHOP) [7].

4. PLANNING FRAMEWORK OF THE ROVER II CONTEXT-AWARE MIDDLEWARE

The Rover II context-aware middleware [2, 6] is a generic middleware, which serves as an integration platform for mobile and desktop applications. It can store and retrieve contextual information, as well as learn and store user behavior models. Figure 1 shows the architecture of Rover II context-aware middleware It consists of several components including a main Controller module (which controls the flow of information among the various components), a Learning Engine (which learns patterns from user's behavior in order to predict a user's intent or goal), a Relevant Information Discovery and Ranking Engine (which determines what information will be relevant to the user's current situation), an Activity Store (which defines what activities the system can perform on the user's behalf) and a Planning algorithm (which generates the sequence of activities that should be performed in order to accomplish a task). A complete description of this system and its architecture is beyond the scope of this paper. Here, we focus on the planning framework of Rover II. This framework (as shown in Figure 1) consists of two components:

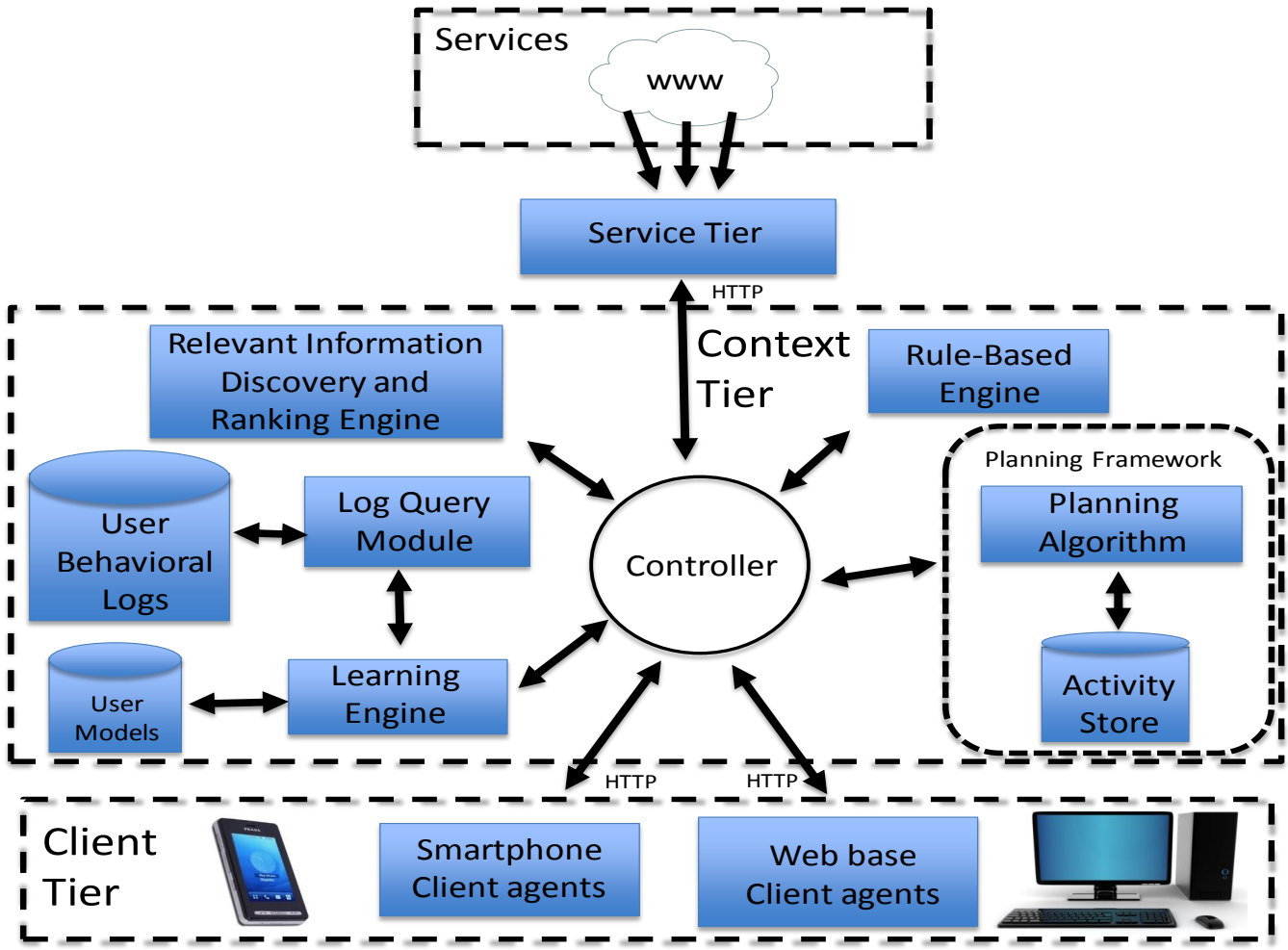

Figure 1: Architecture of the Rover II context-aware middleware

4.1 Planning Algorithm

We employ the Pyhop HTN planning algorithm[1], which is a Python implementation of SHOP [7]. Pyhop is a HTN planner that uses hierarchical decomposition of tasks for planning. It is widely used in hundreds of projects worldwide with applications in industry, academia and government labs. Like other HTN planners, Pyhop is configurable i.e. its planning engine is domain-independent, but the HTN methods may be domain-specific, and the planner can be customized to work in different problem domains by giving it different sets of HTN methods. As mentioned earlier, this ability to use domain-specific problem-solving knowledge can dramatically improve a planner's performance, and sometimes make the difference between solving a problem in exponential time and solving it in polynomial time.

In Pyhop, a task is a symbolic representation of an activity to be performed in the real world, for instance, 'Book a flight'. Instead of spending time on each individual operator, Pyhop uses its in-built hierarchical structure to avoid exponential explosion. Rather than searching through the entire state-space to find the plan, it aims at performing certain tasks that meets predefined conditions. As commonly done in HTN Planning, Pyhop uses abstract tasks to start a plan and then decomposes them into smaller sub tasks. A task can be primitive or non-primitive. A primitive task corresponds to a basic action that can be directly performed in the real world. On the other hand, a non-primitive or compound task is composed of other primitive tasks and cannot be performed directly to the real world. It first needs to be decomposed into simpler tasks until primitive tasks are found.

Algorithm 1 shows the pseudo code for the Pyhop HTN Planning algorithm. It takes the following as input:

- An initial state - This is a description of the current situation.
- List of tasks - These describe the activities to perform
- Methods - These are parameterized descriptions of possible ways to perform a compound task by performing a collection of subtasks. There may be more than one method for the same task.
- Operators - These are parameterized descriptions of what the primitive actions can achieve.

The Pyhop algorithm makes use of backtracking. Backtracking is a general algorithm for finding all (or some) solutions to a computational problem, that incrementally builds candidates to the solutions, and abandons each partial candidate c ("backtracks") as soon as it determines that c cannot possibly lead to a valid solution.

The algorithm recursively checks if it can find a plan for a given set of goal tasks (the planning problem). First it checks if any tasks

[1] https://bitbucket.org/dananau/pyhop

```
Input: Initial state S0, list of tasks T, methods M and operators O
Output: Plan P
Initialize tasklist T to contain the toplevel task in the hierarchy;
Initialize plan P = ∅;
Initialize state S = S0;
  procedure SEEKPLAN(state S, tasklist T, plan P)
    if T = ∅ then
      return plan;
    else
      Task t ←first task from tasklist;
      if t ∈ O then
        if S satisfies the pre conditions of t then
          S' ← t applied to S;
          T ← T - {t};
          P ← P + {t};
          SeekPlan(S', T, P);
          return P (if found);
        else
          return failure;
      else if t ∈ M then
        foreach relevant method do
          τ ← subtasks of t;
          SeekPlan(S, τ + T - {t}, P) ;
          return P (if found);
        end
      else
        return failure;
return Plan;
```

Algorithm 1: HTN Planning algorithm

are left. It is done when no tasks are left, but needs to continue planning if any tasks remain. It then selects the next task and checks if an operator matches the task. If no operator matches the task, the planner looks at the methods. There can be multiple HTN methods to accomplish the same task. If the planner found either an operator or method for a given task, and they did not fail (e.g. when preconditions are not met), the search method is called again for the next task (and thus it is recursive) until a full plan is either found or not. If no plan is found, failure is returned.

4.2 Activity Store containing Domain Description

As mentioned in Section 3, HTN Planning requires domain knowledge. This domain knowledge is specified in our Planning Framework in the form of a domain description consisting of tasks, methods, and operators. The tasks are specified in the form of activities that needs to be performed on the user's behalf such as 'Book Movie'.

The simplest version of a method has three parts: the task for which the method is to be used, the precondition that the current state must satisfy in order for the method to be applicable, and the subtasks that need to be accomplished in order to accomplish that task. For instance, one method to accomplish the task 'Book Movie' for a user is 'Book movie via smartphone app'. This involves sub tasks such as 'searching for the desired movie', 'checking availability of desired date and time', 'checking availability of the required number of seats', 'booking the seats' and 'paying for the tickets'.

Each operator indicates how a primitive task can be performed. Each operator description includes the operator's name and a list of parameters, a precondition expression indicating what should be true in the current state in order for the operator to be applicable and the effects of the operator on the current state if it is applied. For the above mentioned example, the primitive task for 'checking for seats availability' would involve checking if the number of available seats for the desired movie is $>$ the number of seats required by the user.

5. IMPLEMENTATION AND USE CASES

In this section, we present the implementation of our Planning Framework (developed as part of the Rover II context-aware middleware) and its components using specific technologies. The Pyhop planning algorithm is implemented in Python. We have implemented the Activity Store containing domain knowledge as a python module which contains different methods and operators to achieve tasks on the user's behalf. This framework has been integrated with the Rover II middleware using Jython[2], which is an implementation of Python seamlessly integrated with the Java platform.

Some sample use cases that we have implemented are:

5.1 Communicating with a user's contact

Recall the scenario mentioned in Section 2. The system can communicate with the colleague of the user via two possible media: SMS and Email. This use case is implemented as follows:

5.1.1 Task

The high-level task that needs to be achieved in this use case is 'Communicate'.

[2] `http://www.jython.org/`

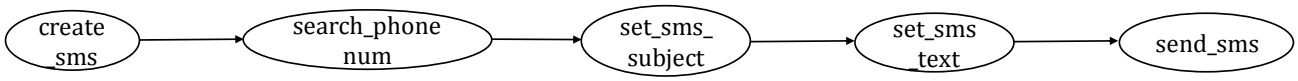

Figure 2: Plan generated by the Planning framework for communicating with the user's colleague

5.1.2 Methods

There are two methods to achieve this task:by SMS or by email.

```
def communicate _by _sms (state, sender, recipient, subject, text, status):
    if ((sender in state.contacts) and (recipient in state.contacts)):
    return [('create _sms', sender, recipient), ('search_phonenum', sender, recipient), ('set _sms _subject', subject), 'set _sms _text',text), ('send _sms', status)]
return False

def communicate _by _email (state,sender,recipient,subject,text,status):
    if ((sender not in state.contacts) or (recipient not in state.contacts)) and ((sender in state.emaillist) and recipient in state.emaillist)):
    return [('create _email', sender, recipient), ('search _emailadd', sender, recipient), ('set _email _subject', subject), ('set _email _text', text), ('send _email', status)]
return False
```

As shown, if the phone number of both the sender i.e. the user and the recipient i.e. his colleague are in the contacts list, the system would send an SMS to the colleague. Otherwise, it would send an email.

5.1.3 Operators

The method to send an SMS (communicate _by _sms) can be decomposed into the following primitive sub tasks or operators:

```
def create _sms(state, sender, recipient):
    if not sender or not recipient:
        return False
    else:
        state.sms['sendername']=sender
        state.sms['recipientname'] = recipient
        return state

def search _phonenum(state, sender, recipient):
    state.sms['senderphone']=state.contacts[sender]
    state.sms['recipientphone'] = state.contacts[recipient]
    return state

def set _sms _subject(state, subject):
    if not subject:
        return False
    else:
        state.sms['subject'] = subject
        return state

def set _sms _text(state, text):
    if not text:
        return False
    else:
        state.sms['text'] = text
        return state

def send _sms(state, status):
    state.sms['status'] = status
    return state
```

Similarly, the method to send an email (communicate _by _email) can be decomposed into the following primitive sub tasks such as:

```
def create _email(state, sender, recipient):
    if not sender or not recipient:
        return False
    else:
        state.email['sendername']=sender
        state.email['recipientname'] = recipient
        return state

def search _emailadd(state, sender, recipient):
    state.email['senderphone']=state.emaillist[sender]
    state.email['recipientphone'] = state.emaillist[recipient]
    return state

def set _email _subject(state, subject):
    if not subject:
        return False
    else:
        state.email['subject'] = subject
        return state

def set _email _text(state, text):
    if not text:
        return False
    else:
        state.email['text'] = text
        return state

def send _email(state, status):
    state.email['status'] = status
    return state
```

These primitive subtasks involve creating the SMS or email, searching for both the sender's and recipient's phone numbers or email addresses, setting the SMS or email subject and text and finally, sending it to the recipient.

5.1.4 Initial state

The initial state to the Planning algorithm includes the contacts and the email list of the user, as well as the subject and text of the message.

5.1.5 Generated plan

The planning algorithm uses the defined domain knowledge (consisting of methods and operators) and current context to generate the appropriate and feasible sequence of actions that need to to be executed in order to accomplish this task. Figure 2 shows a sample generated plan in the case where the phone numbers of both the sender and recipient are present in the contacts list of the user. For legibility, the parameters to these operators have not been shown.

5.2 Booking a movie

We now consider another scenario where the context-aware system needs to book movie tickets for the user. The system may detect from a user's calendar that he/she desires to watch a particular movie on a certain day and after a certain time, say Friday evening. It then proceeds to perform the task of booking the movie proactively. It could also be the case that the user explicitly requests the system to book the movie tickets for a particular date and time.

5.2.1 Task

The high-level task that needs to be achieved in this use case is 'Book a movie'.

5.2.2 Methods

There can be two ways of achieving this task: booking the movie via an app or via a website. The methods representing these two ways are:

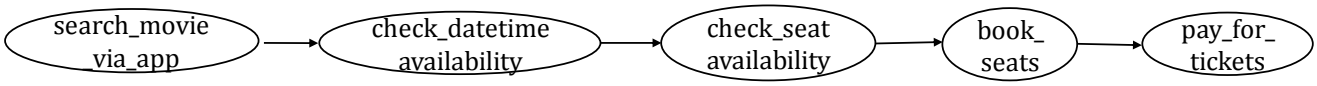

Figure 3: Plan generated by the Planning framework for booking a movie for the user

```
def book _movie _through _app(state,a,movie,dateandtime, seats):
    if ('Fandango' in state.apps):
    return [('search _movie _via _app', a, movie),('check _datetimeavailability',
movie, dateandtime),('check _seatavailability', movie, seats),('book _seats', a, movie,
seats),('pay _for _tickets', a, movie)]
    return False

def book _movie _through _website(state,a,movie,dateandtime, seats):
    if ('Fandango' not in state.apps):
    return [('search _movie _via _browser', a, movie),('check _datetimeavailabil-
ity', movie, dateandtime),('check _seatavailability', movie, seats),('book _seats', a,
movie, seats),('pay _for _tickets', a, movie)]
    return False
```

5.2.3 Operators

These methods can be decomposed into the following primitive sub tasks or operators:

```
def movie _tickets _cost(state,movie,seats):
    return state.ticket[movie]*seats

def search _movie _via _app(state,a,movie):
    if (movie not in state.movies['Fandango'] or state.ticket[movie]>state.money[a]):
        return False
    else:
        return state

def search _movie _via _browser(state,a,movie):
    if (movie not in state.theater['AMC'] or state.ticket[movie]>state.money[a]):
        return False
    else:
        return state

def check _datetimeavailability(state,movie,dateandtime):
    if state.moviedatetimes[movie]>dateandtime:
        return state
    else:
        return False

def check _seatavailability(state,movie,seats):
    if state.movieseats[movie]>seats:
        return state
    else:
        return False

def book _seats(state,a,movie,seats):
    state.movieseats[movie] = state.movieseats[movie] - seats
    state.owe[a] = movie _tickets _cost(state,movie,seats)
    return state

def pay _for _tickets(state,a,movie):
    if state.money[a] >= state.owe[a]:
        state.money[a] = state.money[a] - state.owe[a]
        state.owe[a] = 0
        state.booking[movie]='Booked'
        return state
    else: return False
```

5.2.4 Initial State

The initial state to the Planning algorithm includes the user's current location, movies showing in theaters near the user's current location, the scheduled dates and times of their shows, number of seats available, and the cost of their tickets. An important point to note here is that even though this is domain knowledge, the information is extracted from external sources such as web search engines or other websites. In addition, the user request must contain the name of the movie, the preferred date and time and the number of tickets to be booked.

5.2.5 Generated Plan

The planning algorithm uses the defined domain knowledge (consisting of methods and operators) and current context to generate the appropriate and feasible sequence of actions that need to to be executed in order to accomplish this task. Figure 3 shows a sample plan generated by our framework for booking a movie on the user's behalf.

6. ADVANTAGES OF THE PLANNING FRAMEWORK

Our planning framework offers a number of advantages:

- Minimal user intervention - The framework automatically generates a sequence of actions required to achieve a user's task based on available information and resources without requiring any user intervention. Human guidance is required only at the crucial step when some information is needed from the user. Thus, the users do not have to worry about knowing how exactly to perform certain kinds of tasks in a ubiquitous system, what kinds of services and applications are present in the system and how to interact with them. They can leave the intricate details of performing tasks as well as handling failures to the planning framework.

- Fault-tolerance - If an action fails, the framework detects it and backtracks by retrying actions or by replanning and taking another path to achieve the same goal. The replanning approach to failure recovery works because ubiquitous environments are dynamic and constantly changing. Moreover, there are usually several ways of achieving a task.

- Adaptable to varying context - The planning framework generates the plan to achieve a task taking into consideration the current context of the user and environment. It finds out what information and resources are currently available in the system and then tries to achieve the goal using these resources.

7. RELATED WORK

While AI Planning has been successfully applied in several domains such as robotics and games, it has not been employed in context-aware systems or ubiquitous systems. Most of the existing work in this domain has focused on web services composition using AI Planning [3, 8, 13, 15, 16].

Ranganathan et al. [9] developed a STRIPS-based planning framework that used state space planning for a meeting room domain. The framework was based on a predicate model of pervasive computing where the state of the environment and its elements were represented as a set of predicates. The users were allowed to specify the goals that had to be achieved such as starting a presentation. The actions are either an invocation of a method on a service, device or application and were represented in terms of their preconditions and effects. A utility function is used to determine the best goal state.

8. CONCLUSION AND FUTURE WORK

In this paper, we addressed a key capability required by a context-aware system to act proactively - acting autonomously without an explicit user request. To address it, we proposed a new paradigm for enabling proactivity in context-aware middleware systems by means of HTN planning. We presented the design of a Planning Framework within the infrastructure of our intelligent context-aware middleware called Rover II. We implemented this framework and evaluated its utility with several use cases. We also highlighted the benefits of using such a framework in dynamic ubiquitous systems.

In the future, we plan to carry out further experiments to validate the use of our framework and approach as well as test the scalability of the system in lager environments. We also plan to build a pipeline from our context-aware middleware to client agent applications running on devices such as smartphones or desktops which will enable the plan to be executed on the user's device itself rather than remotely.

9. REFERENCES

[1] P. Bhargava and A. Agrawala. Modeling users' behavior from large scale smartphone data collection. In *under review*.

[2] P. Bhargava, S. Krishnamoorthy, and A. Agrawala. An ontological context model for representing a situation and the design of an intelligent context-aware middleware. In *Proceedings of the ACM Conference on Ubiquitous Computing*, 2012.

[3] J. Bidot, C. Goumopoulos, and I. Calemis. Using ai planning and late binding for managing service workflows in intelligent environments. In *Pervasive Computing and Communications (PerCom), 2011 IEEE International Conference on*, pages 156–163. IEEE, 2011.

[4] K. Erol. Hierarchical task network planning: formalization, analysis, and implementation. 1996.

[5] M. Ghallab, D. Nau, and P. Traverso. *Automated planning: theory & practice*. Elsevier, 2004.

[6] S. Krishnamoorthy, P. Bhargava, M. Mah, and A. Agrawala. Representing and managing the context of a situation. *The Computer Journal*, 55(8):1005–1019, 2012.

[7] D. Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila. Shop: Simple hierarchical ordered planner. In *Proceedings of the 16th international joint conference on Artificial intelligence-Volume 2*, pages 968–973. Morgan Kaufmann Publishers Inc., 1999.

[8] A. Qasem, J. Heflin, and H. Muñoz-Avila. Efficient source discovery and service composition for ubiquitous computing environments. In *Workshop on Semantic Web Technology for Mobile and Ubiquitous Applications (ISWC)*, 2004.

[9] A. Ranganathan and R. H. Campbell. Autonomic pervasive computing based on planning. In *Autonomic Computing, 2004. Proceedings. International Conference on*, pages 80–87. IEEE, 2004.

[10] E. D. Sacerdoti. The nonlinear nature of plans. Technical report, DTIC Document, 1975.

[11] A. Salovaara and A. Oulasvirta. Six modes of proactive resource management: a user-centric typology for proactive behaviors. In *Proceedings of the third Nordic conference on Human-computer interaction*, pages 57–60. ACM, 2004.

[12] M. Satyanarayanan. Pervasive computing: Vision and challenges. *Personal Communications, IEEE*, 8(4):10–17, 2001.

[13] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau. Htn planning for web service composition using shop2. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(4):377–396, 2004.

[14] D. Tennenhouse. Proactive computing. *Communications of the ACM*, 43(5):43–50, 2000.

[15] M. Vukovic and P. Robinson. Adaptive, planning based, web service composition for context awareness. In *In Proceedings of the Second International Conference on Pervasive Computing*, 2004.

[16] M. Vukovic and P. Robinson. Shop2 and tlplan for proactive service composition. In *UK-Russia Workshop on Proactive Computing*. Citeseer, 2005.

[17] R. Want, T. Pering, and D. Tennenhouse. Comparing autonomic and proactive computing. *IBM Systems Journal*, 42(1):129–135, 2003.

[18] D. E. Wilkins. Can ai planners solve practical problems? *Computational intelligence*, 6(4):232–246, 1990.