2011 IEEE International Conference on Pervasive Computing and Communications (PerCom), Seattle (March 21-25, 2011)

Using AI Planning and Late Binding for Managing Service Workflows in Intelligent Environments

Julien Bidot
Institute of Artificial Intelligence
University of Ulm
Ulm, Germany
julien.bidot@uni-ulm.de

Christos Goumopoulos[1,2] and Ioannis Calemis[1]
Research Academic Computer Technology Institute,
[1]DAISy Research Unit, Patras, Greece
[2]ICSE Dept., Aegean University, Samos, Greece
goumop, calemis@cti.gr

***Abstract*—In this paper, we present an approach to aggregating and using devices that support the everyday life of human users in ambient intelligence environments. These execution environments are complex and changing over time, since the devices of the environments are numerous and heterogeneous, and they may appear or disappear at any time. In order to appropriately adapt the ambient system to a user's needs, we adopt a service-oriented approach; i.e., devices provide services that reflect their capabilities. The orchestration of the devices is actually realized with the help of Artificial Intelligence planning techniques and dynamic service binding. At design time, (i) a planning problem is created that consists of the user's goal to be achieved and the services currently offered by the intelligent environment, (ii) the planning problem is then solved using Hierarchical Task Network and Partial-Order Causal-Link planning techniques, (iii) and from the planning decisions taken to find solution plans, abstract service workflows are automatically generated. At run time, the abstract services are dynamically bound to devices that are actually present in the environment. Adaptation of the workflow instantiation is possible due to the late binding mechanism employed. The paper depicts the architecture of our system. It also describes the modeling and the life cycle of the workflows. We discuss the advantages and the limit of our approach with respect to related work and give specific details about implementation. We present some experimental results that validate our system in a real-world application scenario.**

Hierarchical Task Network planning; Partial-Order Causal-Link planning, adaptive workflows; dynamic service binding; ambient intelligence; ontology; services composition framework

I. INTRODUCTION

Intelligent environments (IE), such as smart homes, offices and public spaces, feature a large number of devices and services that help users efficiently perform various kinds of tasks. Combining existing services in pervasive computing environments to create new composite services is in line with the Service-Oriented Architecture (SOA) paradigm [1] and involves special design considerations, including context awareness, adaptation management, device heterogeneity, and user empowerment [2]. In many respects, a composite service can be modeled as a workflow [3]. The definition of a composite service includes a set of atomic services together with the control and data flow among the services. Similarly, a workflow is the automation of a business process, in whole or part, during which documents, information, or tasks are passed from one participant to another for action, according to a set of procedural rules [4]. Workflows have been used to model repeatable tasks or operations in a number of different industries including manufacturing and software. In recent years, workflows have increasingly used distributed resources and Web services through resource models such as grid and cloud computing.

The problem we address in the paper is related to but different from the automated composition of Web services [5]. In addition, we do not focus our work on workflow management systems where workflows can be edited and modified by business people at run time [6].

In this paper, we argue that workflows can be used to model how various services should interact with one another as well as with the user in IEs depending on available resources, environment characteristics, user's tasks and profile. We propose a new approach to orchestrating devices in intelligent environments. Achieving a user's goal with the help of smart devices reverts actually to making (1) strategic decisions; i.e., what basic tasks are to be executed and in which order they are to be executed; (2) operational decisions; i.e., what devices should execute the tasks. Describing the intelligent environment with ontologies allows us to decompose the orchestration problem into two parts: (1) a component integrating Artificial Intelligence (AI) planning techniques responsible for making the strategic decisions at design time in order to create workflows with abstract services, and (2) a component in charge of making the operational decisions at run time in order to generate executable service workflows.

The remaining of the paper is organized as follows. The next section presents some background of the work; the following section lists some related work and compares it to our work; Section IV is devoted to AI planning; the life cycle of workflows is explained in Section V; specific implementation issues and experiments are reported in Section VI; the last section concludes and presents some future work.

II. BACKGROUND

A. ATRACO Project

In the context of the EU funded R&D project ATRACO, we are developing a conceptual framework and a system architecture that supports the realization of adaptive and trusted ambient intelligent systems [7]. In ATRACO, we propose a combination of the SOA model with agents and ontologies (Fig. 1). We have defined the concept of an *Activity Sphere* (*AS*), to be both the model and the realization of the set of information, knowledge, services and other resources required to achieve an individual goal within an IE. The ATRACO approach adopts a unique standpoint in modeling and realizing ASs. We assume that various IEs are already available; each of them hosting a dynamically changing set of heterogeneous and closed smart objects and components. They, nevertheless, contain heterogeneous descriptions of their capabilities and services that can only be accessed from but not modified by other components. Thus, these objects can collaborate in the realization of ubiquitous computing applications within the hosting IE, but the structure of these applications may dynamically change and their efficient operation depends on the orchestration of heterogeneous services. In order to achieve task-based collaboration amongst them, one has to deal with this heterogeneity, while at the same time achieving independence between a task description and its respective realization within a specific IE.

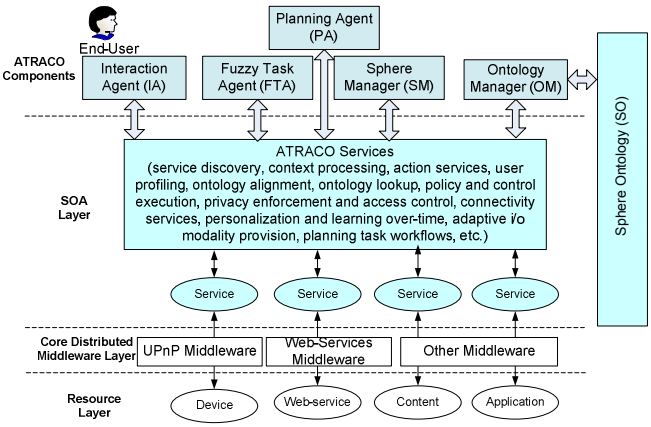

Figure 1. ATRACO architecture.

The ATRACO architecture consists of ontologies, active entities, passive entities, and the user who as the occupant of the IE is at the centre of each AS. Active entities are agents and managers. The role of the ATRACO agents is to provide task planning (Planning Agent or PA), adaptive task realization (Fuzzy systems based Task Agent or FTA) and adaptive human-machine interaction (Interaction Agent or IA). The PA encapsulates a search engine that exploits Hierarchical Task Network (HTN) and Partial-Order Causal-Link (POCL) planning to select atomic services that form a composite service (workflow). One or more FTAs oversee the realization of given tasks within a given IE. These agents are able to learn the user behavior and model it by monitoring the user actions. The agents then create fuzzy-based linguistic models which could be evolved and adapted online in a life learning mode [8]. The IA provides a multimodal front end to the user. Depending on a local ontology, it optimizes task-related dialogue for the specific situation and user [9]. In addition, ontologies complement agents regarding adaptation by tackling the semantic heterogeneity that arises in IEs by using ontology alignment mechanisms to generate the so-called, Sphere Ontology (SO). There are two main kinds of ontologies: local ontologies, which are provided by both active and passive entities and encode their state, properties, capabilities, and services, and the SO, which serves as the core of an AS by representing the combined knowledge of all entities [10].

In this paper, we focus only on the operation of three of the components in ATRACO architecture i.e., PA, SM and OM which support the life cycle of service workflows associated with a user goal.

B. Application Scenario

In order to test our framework and to illustrate how workflows can be planned, executed and adapted to fit user interaction with an IE, we give a simple scenario. This example corresponds to an AS that supports the realization of a goal named "Feel comfortable upon arrival at home."

Martha, a journalist, arrives at the door of her smart apartment. The system recognizes her through an RFID card, and opens the door. On entering the space, the lights and A/C are switched on and brightness and temperature are automatically adjusted according to her profile, season, and time of day, to make her feel comfortable. Martha then sits on the sofa to relax and after a while, the system asks "Would you also like some music?" Martha responds positively, and the music plays (according to predetermined preferences). Following this, the system asks "Would you like to view yesterday's party photos?" Martha responds positively, and a rolling slide show appears in a picture frame in front of her. After a while, Martha gets up, walks towards the window and opens it. Fresh air pours into the room. Temperature level drops. Brightness level increases. Some of the lights are automatically switched off, in an attempt to maintain the previous level of brightness in the room. After a while, the A/C is switched off because of the open window. Suddenly, the picture frame goes off! The system finds a proper replacement and as a result, photos are displayed in the TV set, while Martha is informed about the event. After a period of time, Martha decides to go in the bedroom to relax. The lights and the TV set in the living room go off, whereas some of the lights in the bedroom go on automatically.

The scenario involves the following concepts of ATRACO:

- **AI Planning:** At design time, the ATRACO system needs to find the required abstract services and tasks, and with the use of the PA to provide an adequate abstract service workflow for the above case.
- **Dynamic Service Binding:** At run time, the system considers the abstract service workflow that the PA has produced, and it tries to bind the abstract services to

concrete devices and distribute the workload of the tasks to the various agents (IA, FTA) as appropriate.

- **Location adaptation:** As the user changes location at run time, the system tries to rebind the abstract services to the available devices of the new location.

III. Related Work

The composition of services has been a hot topic in research in the last years, and various AI planning approaches were used to address this issue [11].

The work of R-Moreno et al. [6] integrated AI planning and scheduling techniques to automatically generate business process models, avoiding going through all the drawing process, and making sure that the established connections among activities conform to a valid sequence of activities. After the models have been generated, the user can simulate and optimize the process. The authors use POCL planning and constraint propagation techniques to address this problem. Unlike our work, they implemented a system that can deal with explicit resource and time constraints.

Like the work of Sirin et al. [5], we use HTN planning techniques to model and solve planning problems. However, our implementation is more flexible than theirs, and our planning domain models do not contain any control structure for guiding search in contrast to theirs. In their application, Web services are specified with OWL-S.

Marquardt and Uhrmacher focus on using AI planning to solve the problem of service composition in smart environments [12]. They compare the runtime performance of four different planners using an abstract simulation model, and the evaluation results showed that some of them are suitable for composing services in time. Their modeling of the problem is different from ours, and none of the evaluated planners implements HTN planning. In addition, the service composition is completely done at design time, which makes replanning from scratch necessary each time a new device appears or disappears in the smart environment.

Amigoni et al. proposed an AI planning system, D-HTN, that performs a centralized plan-building activity which is tailored to use the capabilities of the devices currently available in smart environments [13]. In their experimental system, each device is associated with one agent, and another single agent is responsible for building plans. D-HTN implements an HTN planning approach that differs from ours, although their modeling of the problem is similar to ours. D-HTN is not formally grounded and is much more rigid than our search engine. In consequence, D-HTN does not allow for a general backtracking mechanism during the planning process, which is inefficient for addressing complex and large planning problems. An important problem with using D-HTN is that of including knowledge that belongs to the search level into the planning domain models. In contrast to our system, there are no ontologies describing the smart environment, the knowledge about decomposition methods is distributed and collected from the present smart devices, and services composition is entirely performed at design time.

In the Gaia project, Ranganathan and Campbell presented a paradigm for the operation of pervasive computing environment that is based on AI planning [14]. The first difference with our system lies in the modeling of planning problems, since they use PDDL, the initial world state aggregates the state of all entities (i.e. services, devices, and applications) of the environment along with the context of the environment, and the goal world state is determined given a user goal, a template world state, and a utility function to be maximized, but there are no abstract tasks and decomposition methods. In addition, unlike our approach, their planning component is responsible for the binding of concrete devices, devices, and applications, which means that scalability problems inevitably appear due to the very large world states for realistic applications (no abstraction mechanism) and due to the impossibility to declare abstract tasks and procedures in the planning domains (no HTN planning). Their solving procedure becomes very prohibitive, if it has to restart search for solution plans from scratch several times, each time with a different goal world state. Finally, their planning component is also in charge of executing plans, and it may re-execute an action or replan when an execution failure occurs.

The work of Paluska et al. focuses on automating high-level implementation decisions in pervasive applications [15]. Their system enables a model in which an application programmer can specify the behavior of an adaptive application as a set of open-ended decision points. Each of these decision points may be satisfied by a set of alternative, competing scripts. The set may be extended at run time without needing to modify or remove any existing scripts. Their approach is hierarchical, since the scripts may contain themselves decision points. In the same vein as our work, but without using AI planning techniques and ontologies, their system is able to bind services and devices at run time.

In the Pervasive Computing field, several service composition frameworks have been also proposed. PCOM [16] is a component model for pervasive computing. Built on top of BASE, a middleware system that allows for dynamically selecting communication protocol stacks, PCOM treats an application as a collection of distributed components, which make their dependencies explicit. If those dependencies are invalidated, PCOM can attempt to automatically adapt by detecting alternatives according to various strategies. This work addresses the adaptation of an existing application with defined dependencies between the components; however, different or new dependencies have to be explicitly programmed by the developer.

Our approach is closely related to Olympus [17] which extends Gaia middleware with a programming model and framework to realize applications in a polymorphic manner. These applications are structured following a Model-View-Controller-Coordinator pattern and are deployed by mapping abstract descriptions to services using hierarchically defined ontologies. This improves the flexibility and fault tolerance characteristics of applications.

Frameworks based on ontologies, such as METEOR-S [18], lack flexible mechanisms for the distribution of information about services, since they require the adoption of shared ontologies that impose the distribution policy.

While we share many features with the above systems, such as automatic discovery and dynamic binding of services

(PCOM, Gaia), polymorphic behaviour of applications (Olympus) and the use of ontologies to access semantic descriptions of services and for reasoning on the appropriateness of a device (Gaia/Olympus, METEOR-S), however, our planning-based services composition framework supports characteristics that are not found in the above systems. Composition and re-composition of the abstract form of an application is automatically generated by the planning process. This means that while in PCOM and Olympus modifications in the model of the application (component tree or MVC model) must be performed by the developer, in our case this is facilitated by using AI planning techniques to produce abstract service workflows in a more structured and efficient way. This allows for the efficient handling of events such as when user's task intention changes drastically. As is the case with Olympus, service failure recovery and adaptation due to mobility, while conforming to the given abstract description, is done at run-time through an ontology-supported service binding mechanism. However, the use of ontology alignment allows our system to accommodate heterogeneous device specifications that may be different at a lexical and structural level with a degree of accuracy. In addition, by modeling agents as services we can incorporate higher-order adaptation semantics into our service composition framework, either in the form of fuzzy-based Task Agents (FTA) which learn user preferences using fuzzy-based linguistic models or as interaction agents.

IV. Artificial Intelligence Planning

In Artificial Intelligence (AI), the *classical planning problem* consists of a set of operators, one initial state, and one goal state. The instance of an operator is called a task. A state is a set of positive literals (i.e., instantiated predicates). The world state can evolve; e.g., it changes when a task is executed, since every executed task has usually an effect on the current world state. The objective is to select and organize tasks in time that allow us to attain the goal state from the initial state. Tasks can be executed in a world state, only if some preconditions hold in this state. Each task is associated with preconditions and effects. A *plan* consists of tasks and temporal and causal relationships between them.

A. Formal Framework

The PA encapsulates a search engine for AI planning. This search engine relies on a formal framework that integrates Partial-Order Causal-Link (POCL) planning and Hierarchical Task Network (HTN) planning [19]. This hybrid planning framework uses an ADL-like representation of states and basic actions (*primitive tasks*). States, preconditions, and effects of tasks are specified through formulae of a fragment of first-order logic. *Abstract tasks* can be refined by so-called *decomposition methods*, which provide *task networks* (*partial plans*) that describe how the corresponding task can be solved. Partial plans may contain abstract and primitive tasks. With that, hierarchies of tasks and associated methods can be used to encode various ways to accomplish an abstract task.

A *domain model* $D = \langle T, DM \rangle$ consists of a set of task schemata T (operators) and a set DM of decomposition methods. A partial plan is a tuple $P = \langle TE, \prec, VC, CL \rangle$, where TE is a set of *task expressions* (plan steps) $te = l{:}t(\tau)$ with t being the task name and $\tau = \tau_1, \tau_2, \ldots, \tau_n$ the task parameters; the label l is used to uniquely identify the steps of the plan. $\prec$ is a set of *ordering constraints* that impose a partial order on plan steps of TE. VC are *variable constraints* i.e. co-designation and non-co-designation constraints on task parameters. Moreover, VC contains sort restrictions that restrict further co-designations. CL are *causal links* and provide the usual means to establish and maintain causal relationships among the tasks in a partial plan. A causal link $\langle te_i, \phi, te_j \rangle$ indicates, that formula ϕ, which is an effect of task te_i, supports (a part of) the precondition of task te_j. A *planning problem* $\pi = \langle D, s_{\text{init}}, s_{\text{goal}}, P_{\text{init}} \rangle$ consists of a domain model D, an initial state, a goal state, and an initial task network P_{init}. The *solution* of a planning problem is obtained by transforming P_{init} stepwise into a partial plan P that meets the following *solution criteria*: (1) all preconditions of the tasks in P are safely supported by causal links; (2) the ordering and variable constraints of P are consistent; (3) all steps in P are primitive tasks; (4) P is executable in s_{init} and generate a state s_{end} such that $s_{\text{goal}} \subseteq s_{\text{end}}$.

On the one hand, the POCL planning process consists in inserting tasks, ordering constraints, and causal links into partial plans until all preconditions are supported by formulae and all causal conflicts disappear. On the other hand, the HTN planning process endeavors to decompose all abstract tasks, until all tasks in the partial plan are primitive.

In our framework, transforming partial plans into their refinements is done by using so-called *plan modifications*. Given a partial plan $P = \langle TE, \prec, VC, CL \rangle$ and domain model D, a plan modification is defined as $m = \langle E^{\oplus}, E^{\ominus} \rangle$, where $E^{\oplus}$ and $E^{\ominus}$ are disjoint sets of elementary additions and deletions of *plan components* over P and D. Consequently, all elements in $E^{\ominus}$ are from TE, $\prec$, VC, or CL, respectively, while $E^{\oplus}$ consists of new plan components. This generic definition makes the changes explicit that a modification imposes on a plan. Applying a modification $m = \langle E^{\oplus}, E^{\ominus} \rangle$ to a plan P returns a plan P' that is obtained from P by adding all components of $E^{\oplus}$ and removing those of $E^{\ominus}$. Hybrid planning distinguishes various classes of plan modifications such as task decomposition, causal link insertion, and task insertion. The formal framework allows for the implementation of a general backtracking mechanism during search, which makes the search complete, if an adequate search strategy is applied: if there exists a solution plan to the planning problem, then it will be found.

Using this framework for the implementation of a search engine that can address real-world planning problems is particularly advantageous, since it allows us to easily encode and efficiently deal with procedural knowledge thanks to decomposition methods and task networks (HTN planning) and to reason about causal relations between tasks (POCL planning). In addition, the solution plans generated are partially ordered, which is desirable for real-world applications where activities are often performed in parallel.

B. *The Planning Agent*

In ATRACO, the PA encapsulates an AI planning system, which is a search engine for hybrid planning that relies on the formal framework presented in the last section. For specifying a planning problem, we need two sources: the planning domain and the planning problem. The planning domain contains the various operators and the alternatives (by means of decomposition methods) to implement each abstract operator. Furthermore, predicates and types of objects of the application domain are represented. The planning problem describes the problem instance to be solved. The description is made up of three parts: the initial world state, the specification of the goal state, and the initial network. In addition, we declare the objects of the problem at hand; e.g., the person being currently in the apartment.

V. LIFE CYCLE OF WORKFLOWS

We have developed a service composition mechanism which includes 3 phases: *task workflow planning* at design time, *dynamic service binding* at run time, and *execution management and control* at run time, as illustrated in Fig. 2.

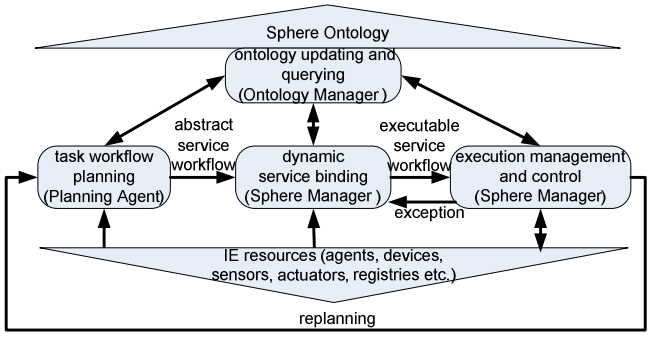

Figure 2. Service composition process in ATRACO.

In ATRACO, we use a library of abstract plans which model specific user's goals. With respect to the user's goals, the PA requests the available services and the user's preferences from the Ontology Manager (OM) in order to create a corresponding planning problem. The planning problem is then solved by the PA. When the search for solution plans is finished, an abstract service workflow is generated from the planning decisions that led to solution plans. An abstract service workflow contains a sequence of abstract services which are actually ontological descriptions of service operations that cannot be directly invoked, but will be resolved by the Sphere Manager (SM) at run time. Having an abstract service workflow description, which is given in a BPEL-like language, the Dynamic Service Binding module of the SM applies a semantic-based discovery mechanism and uses information about available services and context, acquired by the Sphere Ontology (SO) through the OM, to discover suitable services or devices in registries able to perform each abstract service. The output of this process is an executable service workflow. In the execution management and control phase, the SM executes and continuously monitors the deployed services and the termination condition of the workflow.

The OM component provides an interface to the SM to access AS related data, including personal and contextual information, represented in ontologies. The OM provides methods for querying and modifying User Profile Ontologies, Device Ontologies, as well as the eventual SO that emerges from the alignment of all the previous ontologies.

Alignment assumes that in general a mapping can be described as a quadruple $\langle e_1, e_2, n, R \rangle$, where e_1 and e_2 are entities belonging to ontologies O_1 and O_2 respectively, between which a relation is asserted by the mapping; n is a degree of trust (confidence) in that mapping; and R is the relation associated to a mapping. In a nutshell, our alignment solution uses a combination of terminological approaches, which exploit string similarity between labels of ontologies with structural approaches, which rely on the structure of the ontologies. For assessing lexical similarity of concepts and properties, we use the Levenshtein edit distance metric between two strings, enhanced with a synonyms database that we maintain (an external resource such as Wordnet could also be used). However, lexical similarity is not always adequate to verify that concepts are semantically compatible. We, therefore, investigate whether their direct ancestors and descendants as well as sibling entities share lexical similarity. In this way, we can assess the structural relevance of concepts. The approach for the alignment has its limitations. Satisfactory results are obtained when the input ontologies are in the same domain and furthermore non taxonomic relationships (i.e., restrictions) between concepts are not taken into account. The output of the alignment process is an OWL ontology that includes OWL class equivalence assertions that relate the equivalent concepts from the input ontologies. An example of alignment is shown in Fig. 3.

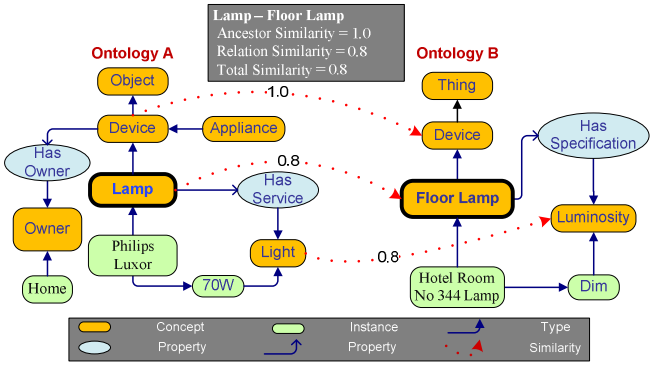

Figure 3. An example of alignment.

A. *Generation of Workflows*

The generation of abstract service workflows is carried out at design time as illustrated in Fig. 4.

1) *Planning Domains and Problems*

Contrary to the approaches presented in the literature [5], we do not create planning domains dynamically. We have enriched the planning domain models with additional constructs that are not used during the planning process but

after it for the automatic generation of abstract service workflows expressed in the BPEL-like language.

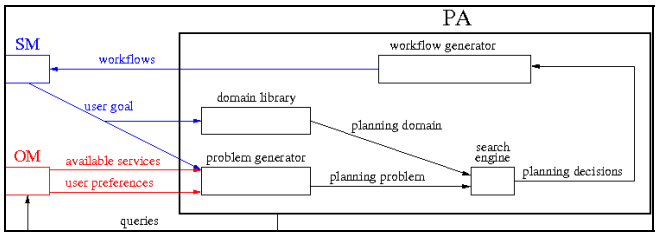

Figure 4. Generation of abstract service workflows.

There are three kinds of basic BPEL activities called respectively `<invoke>`, `<receive>`, and `<reply>`, they are declared as additional attributes of primitive tasks. The structured activities of the BPEL-like language, such as `<while>` and `<repeatUntil>` used to express the repeated execution of activities enrich the decomposition methods. The BPEL constructs `<partnerLink>` used to express the abstract services required and provided by each basic activity are declared as additional attributes of primitive tasks. Each `<partnerLink>` has at least one of two parameters called `myRole` and `partnerRole`, and each construct `<ATRACO:role>` used to associate the abstract services required to fulfill that role is included in planning domain models.

In our application, planning problems are generated dynamically: the currently available abstract services are represented by literals of the initial world state s_{init} (e.g., `luminosity`, `like_music(Martha)`), and the user's goal is expressed by the initial task network P_{init}. For the generation of planning problems, the PA queries the OM and gets from it the list of available services that are relevant for the current planning domain (corresponding to the user's goal); e.g., the PA asks the OM whether there are some devices that can sense the temperature in the room where Martha is living. In addition, the PA queries the OM and gets from it the list of user's preferences with respect to the current planning domain.

2) Planning Process

Since the planning model is designed at an abstract level—the choice of what concrete services and devices to be used is made later by the SM—the decomposition methods describe exhaustively the many ways to decompose a user's goals. As a consequence, the insertion of tasks during backtrack search is not necessary, and the generation of plan modifications of $M_{InsertTask}$ is deactivated during the planning process, which reduces search space drastically. In addition, the goal state of any corresponding planning problem for this application is empty (s_{goal}=Ø).

3) Workflows with Abstract Services

The BPEL construct `<sequence>` used to describe precedence constraints between workflow activities corresponds to an abstract task and the decomposition method that has been applied to decompose it and that describes a network of tasks that are totally ordered with ordering constraints. The construct `<sequence>` also corresponds to any ordering constraint in the solution plan that resulted has been explicitly added during the planning process. The BPEL construct `<flow>` used to describe workflow activities that execute in parallel corresponds to an abstract task and the decomposition method that has been applied to decompose it and that describes a network of tasks that are not ordered.

For the generation of abstract service workflows in our BPEL-like language, the workflow generator of the PA looks for and analyzes the set of plan modifications $M_{success}$ that have led to solution plans and that belong to two classes: $M_{ExpandTask}$ (decomposition of abstract tasks) and $M_{AddOrdConstr}$ (addition of ordering constraints). The formal definition of these plan modification classes is given in [20]. Each plan modification of $M_{ExpandTask}$ is interpreted as a BPEL structured activity such as `<flow>`, `<sequence>`, `<while>`, or `<repeatUntil>` composed possibly of other structure activities and basic activities (`<invoke>`, `<receive>`, and `<reply>`). The analysis of the plan modifications of $M_{AddOrdConstr}$ leads to the identification of `<sequence>` constructs. The information about `<partnerLink>` and `<ATRACO:role>` is directly copied from the planning domain to the abstract service workflow.

B. *Execution and Adaptation of Workflows*

The SM instantiates all the required managers and agents based on the specified workflow and the user reference. In ATRACO, we require adaptive workflows which need to react to varying environmental conditions. This transition from the static to dynamic and adaptive nature of workflows increases the runtime complexity of the management system, since the coordination mechanism must become more fault-tolerant. SM is the ATRACO component responsible for adaptation and execution of the workflows.

In order to do the final binding, SM uses the Device Manager (DM), a module responsible for communicating with the concrete devices. The layer of communication is achieved by using the UPnP protocol. The DM implements a control point and by using the UPnP Simple Service Discovery Protocol, searches for UPnP devices in the local network, and records them to the devices registry. When the SM wants to bind or communicate with a device, it makes the appropriate call to the DM, and the DM by using the General Event Notification Architecture (GENA) event notification protocol, sends the appropriate commands to the devices in the network. Consequently, the DM acts as an intermediate layer between the SM and the final binding to the network devices.

After service binding, the SM starts any interaction task in conjunction with the IA and also any FTA task, and it executes the workflow preserving the precedence constraints or the conditions that are specified in the workflow. At run time, a *Workflow* object aggregates a number of *Task* objects, where each object represents a task in the workflow. The services that this task requires for running are divided into input and output services and are connected with the appropriate resources. The resources that are bound to the

Task object can be either devices that the Task directly controls (i.e., input sensors and actuation devices) or agents, such as the IA or the FTA. In either case, the Task object is informed about the status of the resource and operates according to the pattern specified by its type.

In addition, the SM handles exception events that affect the configuration of the AS. For example, exceptions during the execution of the workflow, such as disconnection or failure of devices trigger an adaptation of the workflow by rebinding services to alternative devices. Context changes during the execution of the workflow may invalidate conditions that were valid during the workflow instantiation. For example, if the user changes location and a follow-me property has been defined for a display service, then the execution state needs to be updated and a new display service instance to be scheduled. In order to achieve workflow adaptation, replanning capabilities may be required by the PA. Replanning comes into play when the dynamic binding fails during workflow execution or update. But, replanning is only necessary in those extreme cases where a lot of similar devices disappear or break down at once, which is very rare.

VI. Implementation Issues and Experiments

In theory, searching for solutions of a planning problem is complex, since the search space associated with such a problem may be huge. For example, using a Pentium 4 processor 3GHz, for a planning domain with 3 abstract tasks, 5 primitive tasks, 8 decomposition methods, 8 predicates, and 2 preconditions and 2 effects per task (midsize number of causal interactions), a corresponding planning problem consisting of two abstract tasks and 13 literals (abstract services) in the initial state is solved after about 6s, after having explored 28 partial plans without backtracking. But the complexity of the backtrack search done by the PA is limited in practice, since (1) the planning domains and the planning problems are designed at a high level: the number abstract services and user's preferences is small; i.e., the world states are small, (2) no task is inserted into the partial plans during the planning process, (3) the number of preconditions and effects per tasks is small, (4) the number of variable constraints is small, and (5) the number of causal interactions between tasks is small. So, for our practical application, the size of search space is small enough to be explored in a few seconds, which is competitive with the experimental results presented in [14].

The search space explored during the planning process is represented by a tree. The cost of generating workflows after the planning process is finished is linear in the number of successful plan modifications $M_{success}$, since all the information we need is stored inside one single path: the cost depends on the depth of the tree (i.e., the number of applied plan modifications) from the initial task network P_{init} to the solution plan.

Considering the dynamic adaptation of the workflows, the main performance issues fall in the binding of the devices to the abstract services. From experiments performed in our IE testbed for the scenario presented in the beginning, a considerable time of the workflow initialization is attributed to querying the devices ontologies. The performance of the OM component is crucial, since a significant burden of the dynamic adaptation is the binding of the most appropriate device for an abstract service, and since many coefficients must be taken into account before selecting a device (location, user, proximity etc.). After running the scenario several times, the following results were produced: (1) 80% of the time for workflow initialization is the time needed for the queries; (2) Queries have a startup load which is linear to the number of devices that exist in the smart space; (3) The average time of workflow initialization is linear to the number of services we need to bind.

In Fig. 5, we see the query time needed for a service to be bound after 15 runs of the experiment in an environment with 35 active devices. We have found that computing multi-criteria numerical conditions required the assertion of extensive number of facts resulting in growing response times. Moving such computations from the knowledge base to an external Java module has improved query times.

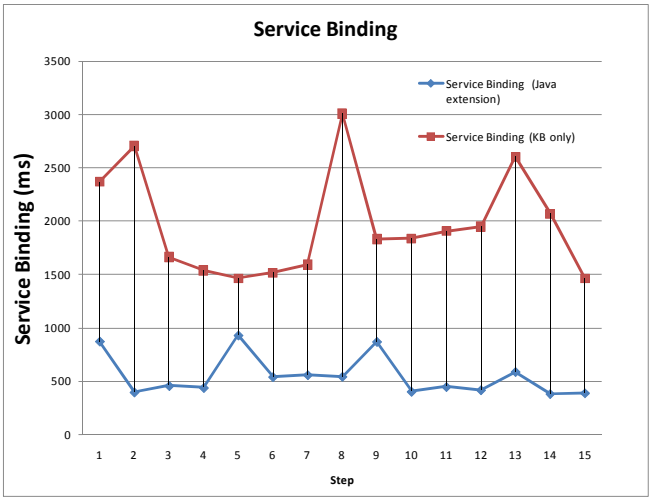

Figure 5. Ontology query time.

The optimized computations provide average time of 550ms, median time of 458ms, maximum time of 930ms, and minimum time of 382ms.

The implementation technologies and tools used are based on open frameworks and are compatible with the SOA paradigm. Java is the main programming language and UPnP enhanced with semantic descriptions [21] is used as the communication middleware for the integration of devices and services, instead of Web services. We use OWL for the development of the ontologies as it provides a strong logical reasoning framework for the expression and enforcement of ATRACO policies and rules. The OM has been developed as a wrapper around the Jena Framework and its interface supports querying of the ontology using SPARQL.

Although there are available (open source) execution engines for BPEL "programs," in ATRACO we need to build a layer upon such engines as a proxy in order to process the parts of the workflow description that are specific to ATRACO. In addition, most engines do not allow for dynamic binding and discovery of services. In order to

address this limitation, the framework uses the SM as a proxy to communicate with service registries to obtain operational descriptions (e.g., UPnP or WSDL files) and instantiate services.

VII. CONCLUSION

Usually, workflow management systems have not been used for dynamic environments requiring automatic adaptive behavior. On the contrary, in ATRACO we require adaptive workflows which need to react to varying environmental conditions. Our general idea is that since a workflow describes the relationship between services and if an agent is represented by such a service, then the relationship between the agents would be possible to specify. Following such a combined agent-based and SOA approach means that a workflow could be used to establish the initial relationships of the multiagent system. Multiagent systems can be specified then first with a workflow description using a BPEL-like language that defines the most common scenario and fault conditions. Once the basic system has been deployed, the agents could be working proactively so they can adapt to unforeseen circumstances and automatically handle the extension to the workflow description. In addition, adaptations of services and devices are possible, since workflows specify abstract services at run time that are bound dynamically. This gives us the opportunity to handle events such as a device failure or using a high-quality service that can replace a service selected in the first place. In that sense, we can see our workflows as adaptive workflows. As our experimental results show, our system is able to deal with a large number of devices thanks to the aggregation and abstraction mechanism. The flexibility offered by abstract service workflows prevents the system from replanning each time a service appears or disappears in the intelligent environment.

As an extension, ATRACO will handle in the near future the simultaneous deployment of multiple workflows in the same space, which may cause synchronization problems where different workflows may compete for the same resources and perform conflicting actions.

ACKNOWLEDGMENT

The research leading to these results has received funding from the EC's 7th FP under grant agreement n° 216837 and from the Transregional Collaborative Research Centre SFB/TRR 62 "Companion-Technology for Cognitive Technical Systems" funded by the German Research Foundation (DFG).

REFERENCES

[1] T. Erl, Service-oriented architecture: concepts, technology, and design. Prentice Hall PTR, 2005.

[2] J. Brønsted, K.M. Hansen, M. Ingstrup, "Service composition issues in pervasive computing," IEEE Pervasive Computing, vol. 9, no. 1, pp. 62-70, 2010.

[3] J. Rao, X. Su, "A survey of automated web service composition methods," LNCS, vol. 3387, Springer-Verlag, pp. 43-54, 2005.

[4] D. Hollingsworth, "The Workflow reference model," Workflow Management Coalition, Document Number TC00-1003, 1995.

[5] 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, vol. 1, no. 4, pp. 377-396, 2004.

[6] M. D. R-Moreno, D. Borrajo, A. Cesta, and A. Oddi, "Integrating planning and scheduling in workflow domains," Expert Systems with Applications, vol. 33, pp. 389-406, 2007.

[7] C. Goumopoulos, et al., "ATRACO: Adaptive and Trusted Ambient Ecologies," Proc. of the 2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO), IEEE CS, pp. 96-101, 2008.

[8] C. Wagner, H. Hagras, "zSlices - towards bridging the gap between interval and general type-2 fuzzy logic," Proc. of the IEEE International Conference of Fuzzy Systems, pp. 489-497, 2008.

[9] G. Pruvost, A. Kameas, T. Heinroth, L. Seremeti, W. Minker, "Combining agents and ontologies to support task-centred interoperability in ambient intelligent environments," Proc. of the 9th International Conference on Intelligent Systems Design and Applications, IEEE CS, pp. 55-60, 2009.

[10] L. Seremeti, C. Goumopoulos and A. Kameas, "Ontology-based modeling of dynamic ubiquitous computing applications as evolving activity spheres," Pervasive and Mobile Computing, vol. 5, no. 5, pp. 574-591, 2009.

[11] D. McDermott, "Estimated-regression planning for interaction with Web services," Proc. 6th Int. Conf. on AI Planning and Scheduling (AIPS'02), pp. 204-211, April 2002.

[12] F. Marquardt and A. M. Uhrmacher, "Evaluating AI planning for service composition in smart environments," Proc. 7th Int. Conf. on Mobile and Ubiquitous Multimedia, December 2008, pp. 48-55.

[13] F. Amigoni, N. Gatti, C. Pinciroli, and M. Roveri, "What Planner for ambient intelligence applications?," IEEE Transations on Systems, Man, and Cybernetics—Part A: Systems and Humans, vol. 35, no. 1, pp. 7-21, 2005.

[14] A. Ranganathan and R. H. Campbell, "Autonomic Pervasive Computing based on Planning," Proc. Of the 1st IEEE Int'l Conf. on Autonomic Computing (ICAC 2004), pp. 80-87, May 2004.

[15] J. Mazzola Paluska, H. Pham, U. Saif, G. Chau, C. Terman, and S. Ward, "Structured Decomposition of Adaptive Applications," Proc. of the 6th IEEE PerCom, IEEE CS, pp. 1-10, 2008.

[16] C. Becker, M. Handte, G. Schiele, K. Rothermel, "PCOM – a component system for pervasive computing", Proc. of the 2nd IEEE PerCom, IEEE CS, pp. 67-76, 2004.

[17] A. Ranganathan, S. Chetan, J. Al-Muhtadi, R. H. Campbell, and M. D. Mickunas, "Olympus: A high-level programming model for pervasive computing environments," Proc. of the 3rd IEEE PerCom, IEEE CS, pp. 7-16, 2005.

[18] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and J. Miller, "METEOR-S WSDI: a scalable P2P infrastructure of registries for semantic publication and discovery of Web services," Information Technology and Management, vol. 6, no. 1, pp. 17-39, 2005.

[19] S. Biundo and B. Schattenberg, "From abstract crisis to concrete relief—a preliminary report on combining state abstraction and HTN planning," Proc. 6th European Conf. on Planning , pp. 157-168, 2001.

[20] B. Schattenberg, A. Weigl, and S. Biundo, "Hybrid planning using flexible strategies," Proc. 28th German Conf. on Artificial Intelligence (KI 2005), pp. 258-272, 2005.

[21] K. Togias, C. Goumopoulos, A. Kameas, "Ontology-based representation of UPnP devices and services for dynamic context-aware ubiquitous computing applications," Proc. of the 3rd Int. Conf. on Communication Theory, Reliability, and Quality of Service (track Models and Ontology-based Design of Protocols, Architectures and Services), IEEE CS CPS, pp. 220-225, 2010.