Chapter 1
A Middleware Architecture for Ambient Adaptive Systems

C. Goumopoulos

Abstract Ambient adaptive systems have to use mechanisms to regulate themselves and change their structure in order to operate efficiently within dynamic ubiquitous computing environments. First of all we outline a survey on existing middleware solutions for building ambient adaptive systems. After, discussing the limitations of the existing approaches, we present our propositions for a middleware architecture to support dynamic adaptation within ambient environments. Our approach is based on the Service-Oriented Architecture (SOA) paradigm which can be considered as an evolution of the component-based design paradigm. The aim is to use component interfaces for the identification and automated connection of components acting as service providers/consumers. The proposed middleware provides a solution that supports the adaptation of applications at the structural level, where the structure of the application can change through dynamic service composition. We call this adaptation 'polymorphism' in analogy with the synonymous term found in the object-oriented programming paradigm. Besides SOA, we use a set of intelligent agents to support adaptive workflow management and task realization based on a dynamically composed ontology of the properties, services and state of the environment resources. An experimental prototype is provided in order to test the middleware developed.

1.1 Introduction

Intelligent environments (IE), like smart homes, offices and public spaces, are featured with a large number of devices and services that help users in performing efficiently various kinds of tasks. Combining existing services in pervasive computing environments to create new distributed applications can be facilitated by middleware architectures, but this should accomodate special design considerations, including

Christos Goumopoulos
Research Academic Computer Technology Institute, Patras, Greece, e-mail: goumop@cti.gr

context awareness, adaptation management, device heterogeneity, and user empowerment [6].

Traditional middleware, such as Remote Procedure Calls [4], OMG CORBA [11], Java Remote Method Invocation (RMI) [44] and Microsoft Distributed Component Object Model (DCOM) [20] facilitate the development of distributed applications and help to resolve problems such as tackling the complexity of programming inter-process communication and the need to support services across heterogeneous platforms. However, traditional middleware is limited in its ability to support adaptation.

Ambient adaptive systems which are a special category of distributed systems operate in a dynamic environment. The dynamicity of the environment may relate with evolving user requirements and varying execution context due to the diversity of available devices, user preferences and services. Consequently there is a need for both applications and infrastructure to be designed for change. The evolution of user requirements calls for system evolution. The dynamic execution environment calls for dynamic adaptation. In order to allow evolution, the internal structure of the system must be made open in order to support proactive and reactive system reconfiguration.

In this work, we present firstly a survey of the state-of-the-art on existing middleware solutions for building adaptive ambient systems. After, discussing the limitations of the existing approaches, we present our propositions for middleware architecture to support dynamic adaptation within ambient environments. Our approach uses the service-oriented architecture paradigm coupled with agents and ontologies. The aim is to use component interfaces for the identification and automated connection of components acting as service providers/consumers. The proposed middleware provides a solution that supports the adaptation of applications at the structural level, where the structure of the application can change through dynamic service binding. Behavioural adaptation, not examined here, is also possible when the application logic is changed as a result of learning. An experimental prototype is provided in order to test the middleware developed.

1.2 Related Work

Three key paradigms that can be used to build adaptive systems are computational reflection, Aspect-Oriented Programming (AOP) and service oriented architectures. Researchers have also explored the possibility to combine different paradigms such as AOP and reflection in middleware systems to increase support for the development of dynamic distributed systems [19]. In the following we examine how each one of these paradigms can support the development of adaptive systems.

1.2.1 Reflective Middleware

In principle *computational reflection* allows a program to observe and modify its own structure and behavior at runtime, by providing a self-representation that can be accessed and changed by the program [31]. More importantly, these changes must be causally reflected to the actual computations performed by the program. In particular, the architecture of reflective systems follows a kind of "white-box" approach that provides comprehensive access in the internal details of a system allowing dealing with highly dynamic environments, for which run time adaptation is required. This is conceptually contrary to the encapsulation principle in objectoriented programming followed by traditional middleware that adopt the remote object model. The reflection technique was used initially in the domain of programming languages as a means to help designing more open and extensible languages. The reflection is applied also in other domains including operating systems and distributed systems. Recently, reflection has been also applied in middleware, which needs to adapt its behavior to changing requirements when operating in a dynamic environment [27]. The dynamic modification of the middleware implementation allows for the adaptation of the behavior of distributed applications that are based on this middleware. Typically, reflective middleware provides adaptation of behavior of distributed applications in terms of non-functional requirements such as QoS, security, performance and fault tolerance.

A reflective system is organized into two levels called *base-level* and *meta-level*. The former represents the basic functionality of the system. The latter models the structural and computational aspects of the base level in order to observe or modify the behavior of the objects that exist in the base level, assuming an object-oriented system. The reflective approach supports the inspection and the adaptation of the underlying implementation (base-level) in run-time. A reflective system provides a *meta-object protocol* (MOP) in order to determine the services that are available in meta-level and their relationship to the base-level objects[26]. The meta-level can be accessed via a process called reification. Reification is the disclosure of certain hidden aspects of the internal representation of the system in terms of programming entities that can be manipulated at runtime. The "opening of" implementation offers a simple mechanism in order to interpose some behavior (e.g., add a method in an object, save the state of the object, check security issues, etc.) with a view to watch or alter the internal behavior of the system.

Reflection enables an application to adjust its behaviour based on a reflective middleware that allows inspecting and adapting its own behavior according to application's needs. Figure 1.1 shows schematically a simple example of this situation. A meta-object (`mObj`) has been defined at the meta-level and is associated through MOP to a base-level object (`Obj`) belonging to some middleware implementation. An application calls a method of the middleware (`Obj.Method()`) which is reified through MOP and a defined object with a specified reference (`ref`). This object is passed to the associated meta-object that executes a method (`mObj.mMethod(ref)`). This method executes a logic specified by the MOP interface and then passes control to the original method through a reflection method (`base_Method(ref)`). The meta-object receives the results, performs any post-processing specified by the MOP and returns to the calling application.

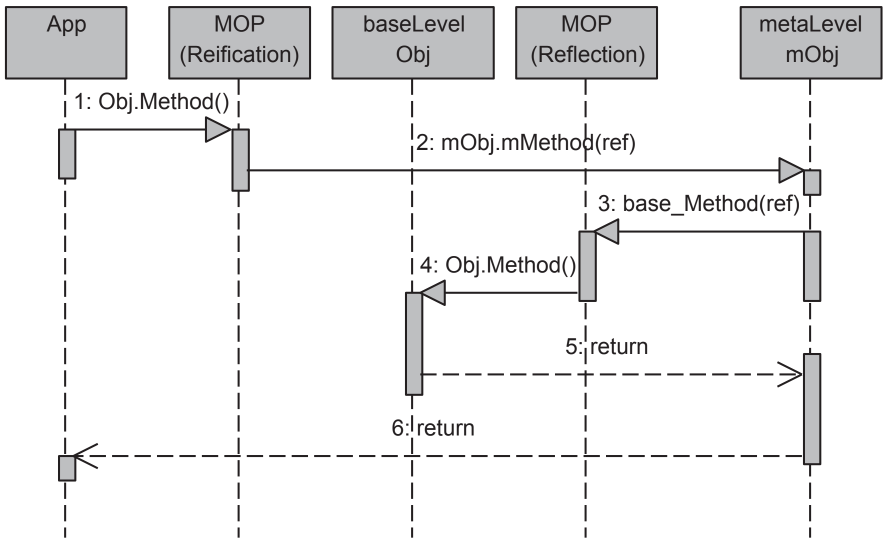

Fig. 1.1 An application calling a method in a reflective middleware.

A category of reflective systems support a higher level reflection in the sense that they can add or remove methods from the objects and classes dynamically and even change the class of an object in run-time. The practical result is to be able to restrict the size of middleware with a minimal total of operations that can run in devices with limited resources. On the contrary, other systems are focused in simpler reflective forms in order to achieve a better performance. Their reflective mechanisms are not part of the normal flow of control and are only called when needed.

Middleware systems that integrate reflection in their architecture have been developed as research prototypes. In the following we cite a few examples of reflective middleware. A number of early systems such as FlexiNet [22], OpenCorba [30], dynamicTAO [28] and OpenORB [5] were based on CORBA and targeted flexibility and dynamic reconfigurability of Object Request Broker (ORB). However, these systems suffered from the heavy computational load imposed by CORBA. Capra et al in [7] discuss CARISMA, which uses reflection to support dynamic adaptation of middleware behaviour to changes in context (e.g., adapting a streaming encoder binding in variable QoS conditions) and ReMMoC, which uses reflection to handle heterogeneity requirements imposed by both applications and underlying device platforms. Both approaches target a minimal reflective middleware for mobile devices where pluggable components can be used by developers to specialize the middleware to suit different devices and environments, thus solving heterogeneity issues. QuA middleware explores the principle of mirror-based reflection to design a reflective API according to the programming abstractions defined by a language

[14]. In the QuA middleware approach a mirror can be defined to reflect a service, in terms of middleware abstractions like type, interface, service and binding, without being dependent the running instances.

Even though reflection is a powerful mechanism to construct adaptive systems there are still issues that need to be understood and solved. The performance of reflective middleware is a matter that is open for further research. The majority of reflective systems impose a rather heavy workload that would cause significant performance deterioration in devices with limited resources and there is always a trade-off issue between performance and scope of adaptability. Another issue that needs to be addressed is dynamically tuning the scope of changes when reconfiguring the system based on adaptation semantic information.

1.2.2 Aspect Oriented Programming

Aspect-Oriented Programming (AOP) [24] is a software development paradigm that emphasizes decomposition of complex programs in terms of intervened cross-cutting aspects, such as QoS, security, persistence, fault-tolerance, logging and resource utilization. This is different from other programming paradigms which emphasize functional decomposition breaking a problem into units like procedures, objects and modules. For instance, object-oriented programming uses inheritance hierarchies to abstract commonalities among classes, however global aspects (affect many classes) are implemented in an ad-hoc manner and become tightly intermixed with classes, which makes changes to the program difficult and error prone. On the other hand, AOP supports the concept of separation of concerns to counter this problem. AOP defines the methods and tools to separate cross-cutting aspects during development time. The source program consists of modules that deal with the different aspects that are described independently. All these modules are integrated during compile (statically) or run time (dynamically) to form a global application with new behaviour using a composition tool called *aspect weaver*.

AOP combines principles of object-oriented programming and computational reflection discussed in the previous section. AOP languages have functionality similar to, but more restricted than meta-object protocols and use a few key concepts: *join points*, *point cuts*, and *advices*. A *join point* is a place, in the source code of the program, where aspect-related code can be inserted. A join point needs to be addressable and understandable by an ordinary programmer to be useful. It should also be stable across typical program changes in order for an aspect to be stable across such changes. Aspect weaving relies on the concept of *point cut*, i.e. the specification of a set of join points according to a given criterion, and *advice*, i.e. the definition of the interaction of the inserted code with the base program. An advice specifies whether the inserted code should be executed before, after, or in replacement for the operations located at the point cuts. Two Java based composition tools that implement the AOP paradigm are AspectJ [25] and JAC [34].

AOP benefits outlined above are important to adaptive middleware. Such approach enables the separation of middleware cross-cutting concerns (e.g., security, logging) at development time and later at compile or run time, where these concerns can be selectively woven into application code. Using AOP, tailored versions of middleware can be generated for application-specific domains. In the following we cite a few examples of adaptive middleware developed based on AOP principles. Yang et al in [45] proposes a systematic approach for preparing an existing program for adaptation and defining dynamic adaptations. The approach uses a static AOP weaver at compile time and reflection during run time. This basic scheme has been followed by others researchers also. Frei et al in [13] present an architecture supporting dynamic AOP that establishes an event infrastructure to extend existing application's behavior at runtime. When the application extension is activated, the dynamic AOP platform inserts an AOP aspect into the AOP platform which intercepts the application's execution and monitors its progress. Whenever the application reaches selected points in the execution, the AOP platform redirects the execution to the appropriate application extension. The memory footprint of the platform is however quite heavy (1MB) to run on resource-constrained devices. Similarly, Maciel da Costa et al in [9] discuss an adaptive middleware architecture, based on aspects, which can be used to develop adaptive mobile applications. A mail server prototype was implemented based on Web Services, Java and AspectJ technologies to evaluate the architecture regarding operation adaptation depending on resource utilization (e.g., power consumption).

AOP has advantages, such as separation of cross-cutting concerns, but presents also difficulties. Programming in terms of aspects requires much more than just identifying the different aspects of concern. It requires being able to express those aspects of concern in a way that is precise and that makes the relations among the aspects of concern precise. This is what enables the aspect weaver to work, and is also what makes possible reasoning about the code or debugging the code. Another important problem related to AOP is the composition of aspects. For instance, if different pieces of aspect-related code are inserted at the same join point, the order of insertion may be relevant if the corresponding aspects are not independent. Such issues cannot usually be settled by the weaver and call for additional specification. Finally, most AOP approaches do not support adequate point cut descriptions to capture join points based on context data and business-level semantics.

1.2.3 Service-Oriented Architecture

The Service Oriented Architecture (SOA) paradigm has been envisioned as an evolution of the component-based engineering paradigm centered on the concept of service [12]. This can be applied in the design of distributed applications that are seen as a composition of services. In addition, the service concept can be applied recursively, since a system component can provide a service, but simultaneously it can encapsulate a composition of services from its service requestors.

In an SOA environment, resources on a network are made available as independent services that can be accessed without knowledge of their underlying platform implementation. A provided service is usually embodied in a set of interfaces, each of which represents an aspect of the service. In general this set contains the operations that a service supports, and some information on how to access these operations. Service interfaces can be published in registries, which also provide services themselves (publish and discovery services), allowing the potential service requestors to discover and access these services (Figure 1.2).

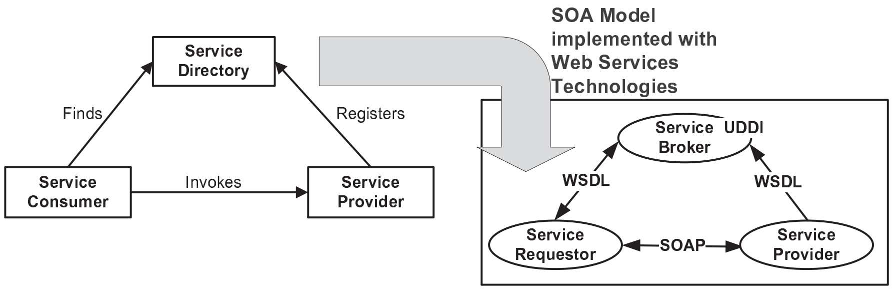

Fig. 1.2 SOA conceptual model.

The independent deployment of services enables late binding which is essential for adaptive systems. *Late binding* provides the opportunity for dynamic composition of services or for swapping two compatible services at run time through a well defined interface. In the SOA paradigm, we can view two abstraction levels of the service concept. *Elementary services* are basic functionalities, usually provided by resources (e.g., devices) in an AmI environment. *Composite services* assemble a set of functionalities in relation to user tasks, and thus are closer to user actual goals. Service composition has widely been addressed in the Web Service field. Existing composition frameworks [8] enable expressing and enacting complex service compositions. However, they rely on explicitly named services, which are not discovered dynamically. On the contrary, the Semantic Web Services (SWS) approach [32] is a step toward dynamic service discovery and composition [40], [10], where intelligent systems try to build composite services from abstract user requirements with or without manual selection of services. SWS leverage knowledge representation techniques, with ontologies describing a domain in a formal manner, and AI planning methods to make composition systems more autonomous.

Although, Web Services are a key implementation technology of the SOA paradigm, the main standards defined to implement the SOA paradigm (i.e., WSDL, UDDI, SOAP) emphasize interoperability rather than the capability to accommodate seamless changes at runtime. Frameworks based on ontologies, such as METEOR-S [42], also lack flexible mechanisms for the distribution of information about services as they require the adoption of shared ontologies that impose the distribution policy. Regarding composition, Business Process Execution Language (BPEL) is the de-facto standard [1]. It takes a workflow-oriented approach to the coordination of cooperating services and provides a good solution for the design-time composition of heterogeneous components wrapped as WSDL services. However, runtime identification of partner services is not addressed and thus the degree of dynamism and flexibility is limited.

1.2.4 Overview

Based on the above discussion we give in the following table an overview of the relative advantages/shortcomings of the three middleware design paradigms regarding their support to the development of adaptive systems.

Table 1.1 Pros and cons of the three middleware design paradigms.

Paradigm	Pros	Cons
Reflective Middleware	• a system can modify its structure and behavior at runtime • achieves variable system size to suit different devices and environments • changes made to the self-representation are immediately mirrored in the underlying system's actual state and behavior (causally connected)	• conceptually contrary to the encapsulation principle • usually heavy computational load and low performance • dynamically tuning the scope of changes based on adaptation semantic information is still an open issue
Aspect Oriented Programming	• supports the concept of separation of concerns • separates cross-cutting aspects during development time (e.g., security, logging) • combines principles of object-oriented programming and computational reflection	• may give large memory footprint • programming in terms of aspects is not easy • the order of insertion may be relevant if the corresponding aspects are not independent (composition of aspects) • does not support adequate point cut descriptions to capture join points based on context data and business-level semantics
Service-Oriented Architecture	• modular design appropriate for adaptation and reconfiguration • late binding of services • composite services can be defined from simple ones • main standards defined to implement SOA provide support for interoperability	• automatic service composition is not trivial • main standards defined to implement SOA provide limited capability to accommodate seamless changes at runtime • limited degree of dynamism and flexibility

In this work, we describe an approach based on the SOA paradigm. Besides SOA a novel mechanism is proposed to achieve different kinds of adaptation centered upon the management of knowledge, which is encoded in multi-layered ontologies, which are used by intelligent agents.

1.3 ATRACO Architecture

Ambient Intelligence (AmI) is a paradigm that puts forward the criteria for the design of the next generation of UbiComp environments [37]. In this context we have introduced the *Ambient Ecology* (AE) metaphor to conceptualize a space populated by connected devices and services that are interrelated with each other, the environment and the people, supporting the users' everyday activities in a meaningful way [16].

In the context of the EU funded R&D project ATRACO [18] we aim to extend the AE concept by developing a conceptual framework and a system architecture that will support the realization of adaptive and trusted AEs which are assembled to support user goals in the form of *Activity Spheres* (ASs). Our approach is based on a number of well established engineering principles, such as the distribution of control and the separation of service interfaces from the service implementation, adopting a SOA model combined with intelligent agents and ontologies. Agents support adaptive planning, task realization and enhanced human-machine interaction while ontologies provide knowledge representation, management of heterogeneity, semantically rich resource discovery and adaptation. ATRACO ASs are dynamic compositions of distributed, loosely-coupled and highly cohesive components that operate in dynamic environments.

Therefore the architecture and the system we propose operate in an AmI environment, which is populated with people and an AE of devices and services. Our basic assumption is that the AE components are all autonomous, in the sense that (a) they have internal models of their properties, capabilities, goals and functions, and (b) these models are proprietary and "closed", that is, (i) they are not expressed in some standard format and (ii) they can only be changed by the owner components. However, each component can be queried and will respond using a standardized protocol.

1.3.1 ATRACO World Model

The concepts discussed below constitute a critical subset of the ATRACO conceptual framework defined for building AmI applications.

The basic terms and concepts of the ATRACO world model are encoded in the ATRACO Upper Level Ontology (ULO). In general, ontology is used as the means to share information among heterogeneous parties in a way that is commonly understood [21]. An ontology is a network of concepts and entities, which can be associated with different types of relations (the most common being the hierarchical association, or is-a relation). More concrete (or domain) ontologies contain also instances of these entities with specific properties and values. More powerful ontologies contain constraints and rules that cause inferences for the entities. Figure 1.3 illustrates in UML representation the AS domain model which is also encoded as ontology in the ATRACO ULO.

Table 1.2 ATRACO main concepts and corresponding descriptions.

Concept	Description
Ambient Ecology (AE)	The set of heterogeneous artefacts with different capabilities and provided services that reside within an Intelligent Environment (IE).
Activity Sphere (AS)	It is formed to support an actor' specific goal. An AS represents 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 concept of AS is a "digitization" of the concept of "bubble" used by the psychologist Robert Sommer [39] to describe a temporary defined space that can limit the information coming into and leaving it.
Intelligent Environment (IE)	A territory that has both physical properties and offers digital services. It is the container of AE. ASs are instantiated in an IE using the resources provided by its AE.
Artefact	A tangible object which bears digitally expressed properties; usually it is an object or device augmented with sensors, actuators, processing, networking unit etc. or a computational device that already has embedded some of the required hardware components.
Actor	Any member of AE capable of setting and attaining goals by realizing activities. Within the AE actors are users or agents.
Goal	Each actor may have its own set of goals and plans to achieve them. A goal is described as a set of abstract tasks, which is described with a task model.
Task Model	It may be *abstract* or *concrete*. An abstract task model describes what should be done, without details of how it should be done or by the use of what kind of modality; these are described in the corresponding concrete model. The abstract task model may also contain several decomposition rules modelled as a set of subtasks.
Local Ontology (LO)	Each member of the AE stores locally descriptions of its properties, services and capabilities. It is a sub-class of the class Ontology.
Sphere Ontology (SO)	The SO results from the LO of those AE members that are required to achieve the AS's goal based on the resolution of its task model. Apart from device and service ontologies, it may contain user profiles, agent rule bases and policies. It is another sub-class of the class Ontology.
Agent	A software module (is a kind of actor) capable of pursuing and realizing plans in order to achieve specific goals based on tasks. It includes three types of agents: *Task Agent* (e.g., Fuzzy systems based Task Agent or FTA), who manipulates sensors and actuators in order to realize specific tasks; *Planning Agent* (PA), who resolves an abstract task hierarchy into concrete tasks using the resources of the AE; and *Interaction Agent* (IA), who manages user-system interaction using a mixed-initiative dialogue model.
User	The actor that uses the available services and devices in order to perform a task. When a user performs a task, this can be subdivided into different activities. Users use devices, which provide them with services. Devices run these services in a physical environment (context). Users use these services according to personal conditions (user profile) and within a physical context.
Aim	It is attributed to a user; it is decomposed into a set of interrelated goals, which are distributed to the components of the AS.
Policy	Actors specify high-level rules for granting and revoking the access rights to and from different services. Examples of policy ontologies are privacy policy ontology, interaction ontology and conflict resolution policy ontology.
Service	The entity which describes the service offered by a device.
Device	The entity that has physical/digital properties and offers a specific service.
Resource	A resource can be the *space*, an *entity*, or a *component*, such as managers (e.g. Ontology Manager, Sphere Manager) or other basic components.

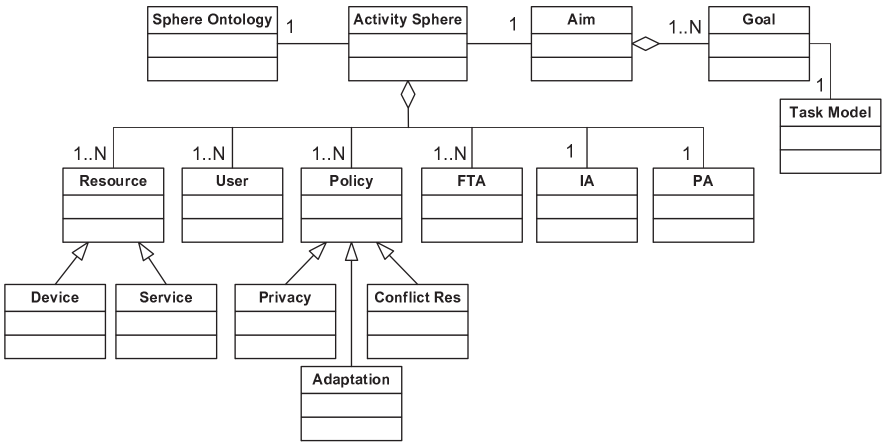

Fig. 1.3 Activity Sphere domain model (part of ATRACO ULO).

1.3.2 System Requirements

For the requirement analysis and design of the ATRACO architecture we followed a process where initially application scenarios were defined and application requirements from a user perspective were identified. In addition as a separate process we defined high-level requirements on system perspective. Then initial requirements were used as input for a process of abstraction that allowed identifying a set of challenges that the architecture has to address, in order to frame further design [18]. These challenges are organized in the following categories:

Challenge 1: Assemble/Dissolve The first challenge has naturally to do with the formation and the dissolution of ATRACO applications (ASs). ASs encapsulate the Ambient Ecology resources that are necessary to serve the goal for which the AS has been created. ATRACO supports adaptation and trust requirements of ASs by integrating into the AS services of the system components that develops.

Challenge 2: Adaptability Adaptability implies that an AS should attempt to continuously provide expected behaviour by adapting to unexpected conditions such as changes to the resources constituting the system or changes in the behavior of the user.

To this effect, the ATRACO system components are defined that adapt task-based usage of the sphere to the changing user behaviour, environment conditions and context. ATRACO implements mechanisms that support adaptability in several forms:

- *activity sphere adaptation*, in terms of *structural adaptation*: the persistent achievement of the goal when changes on the type or cardinality of the available resources occur
- *behavioural adaptation*, where the application logic is changed as a result of changes in the user and/or device behavior. This category is specialized as

  - *artefact adaptation* (the system examines how an artefact can adapt its model of operation in reaction to changes in the device characteristics e.g., handling a partial failure of a heater) and
  - *user behaviour model adaptation* (an agent will learn and adapt its rule base to face the changes in the user desires and preferences by monitoring the user actions e.g., the user decides to read in bed, therefore requires her bedside lamp to be on instead of her reading light).
- *user interaction adaptation* specifies adaptation interacting with the user using different devices/modalities depending on available resources, environment characteristics, tasks and user profile.
- *network adaptation* to allow the uniform and transparent access to devices and services present in the networked environment supporting the realization of activity spheres across a mixture of heterogeneous networks.

Challenge 3: Semantic heterogeneity A basic assumption is that an AmI space is available to host an ambient ecology and devices and services are inherently heterogeneous and contain heterogeneous descriptions of their capabilities and services in the form of local ontologies. Thus, in order to achieve collaboration among them, firstly one has to deal with these forms of heterogeneity. However, the issue raised by the heterogeneity of ontologies and how to achieve semantic interoperability between systems using different ontologies is a challenge. In ATRACO, the approach that is followed in order to address this challenge is to research, develop and test theories of ontology alignment to achieve task-based semantic integration of heterogeneous devices and services.

To this effect, a Sphere Ontology is defined and an Ontology Manager administrates its use by performing ontology updating, ontology querying and ontology matching services.

Challenge 4: Trustworthiness The interactions in the activity sphere should be trustworthy. The ambient ecology will behave in a dependable manner and will not adversely affect information, other components of the system or people.

To this effect, policies and rules are defined in the ontology and mechanisms are defined for the management of the identity of service requestors and service providers as well as access control on services and context information.

Summarizing the above discussion, ATRACO architecture should provide:

- support for realizing user goals (activity spheres), by resolving abstract tasks to a workflow of concrete tasks;
- support for executing workflows by applying service composition and control policies in the form of rules (obligation policies);
- support for establishment and management of associations between service clients and service providers (as described in task workflows);
- support for maintaining the sphere ontology which contains the contextual knowledge necessary to realize the concrete tasks;
- support for ontology alignment and lookup;
- support for adaptation of the given tasks according to the user desires and behaviour (personalization and learning over-time);

- support for use of heterogeneous network capabilities for communication (network adaptation);
- support for discovery of services, devices, networks and resources;
- support for usage of services offered within ATRACO infrastructure or by third parties (e.g., external Web Services);
- support for privacy enforcement and access control through policies;
- support for the possibility of adapting the user interaction depending on available interactive devices and objects;
- support for management of user profiles and preferences;
- support for gathering, processing and distribution of context information;

In the next section we outline the ATRACO system design that accommodate the system requirements and then we discuss in more detail the service composition framework for deploying adaptive workflows in IEs to achieve structural adaptation of ATRACO applications, which is the focus of this presentation.

1.3.3 System Design

In ATRACO, we propose a combination of the SOA model with agents and ontologies (Figure 1.4). We adopt SOA both at the resource level to integrate resources, such as devices, sensors and context in applications and at the system level to combine ATRACO services that provide adaptation and trust features into applications. ATRACO aims to empower users with the ability to interact in environments with many resources such as devices (UPnP devices), web-services, content (music/video file, contacts) and applications (e.g., media player) using adaptive user interfaces. The functionality in these environments is exposed as semantically rich services which an actor (either a user or an agent) can discover and then compose to form ATRACO Activity Spheres.

Each service is associated with at least one semantic description which shields the actor from the complexity of the Resource Layer realization and makes it easy for the actor to employ these services in accomplishing interesting and useful tasks. Figure 1.4 shows a conceptual layered view of the ATRACO architecture. The ATRACO infrastructure consists of SOA services. On the one hand, these context-aware services are built on "Core distributed middleware" and rely on "'Network and resources" layer. On the other hand, the ATRACO infrastructure supports basic services such as context management and reasoning, communication management, user profiling and service (discovery), as well as adaptation and privacy services that form the basis for ATRACO systems (i.e., ASs).

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).

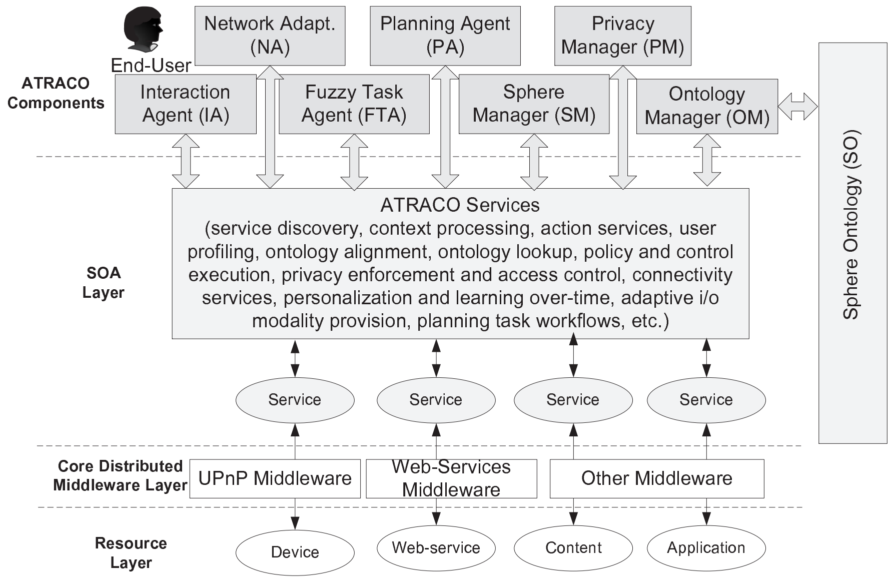

Fig. 1.4 ATRACO architecture.

The PA encapsulates a search engine that exploits hierarchical planning and partial-order causal-link planning to select atomic services that form a composite service (workflow) [3]. 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 [43]. 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 [35]. The IA may be triggered both by the FTA and the PA to retrieve further context information needed to realize and plan tasks by interacting with the user. On the other hand, 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 [38].

The Sphere Manager (SM) and Ontology Manager (OM) components are responsible for the formation, adaptation and evolution of the user applications (modeled in ATRACO as ASs) and will be further examined in this paper. In the current version of the system there is also a Privacy Manager (PM) that provides a set of privacy enhancing techniques in order to support privacy in an adaptive and individualized way. Finaly, devices in the IE that may come from heterogeneous networks (e.g., LonWorks, ZigBee, Z-Wave, etc.) and services (e.g., Network Time, VoIP, Real Time Streaming, etc.) are accessed transparently through a service representation layer exporting them to the ATRACO clients as UPnP services. This layer is implemented in the Network Adaptation (NA) component [33].

1.4 Adaptive Workflows and Structural Adaptation

In many respects, a composite service can be modeled as a workflow [36]. 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 [23]. 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. In this section, 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 tasks and profile.

In this section we describe how SOA can support AS adaptation. The structural adaptation (a form of polymorphism) is possible because the workflow model represents abstract services and binding to real devices can be accomplished at runtime. ATRACO-BPEL, a streamlined version of BPEL, has been defined as the specification language to describe workflows of abstract services.

1.4.1 Scenarios

In order to test our framework and to illustrate how workflows can be used to fit user interaction with an IE, as well as the structural adaptation mechanism of ASs, we use two simple scenarios. The first example corresponds to an AS that supports the realization of goal named *“Feel comfortable upon arrival at home”*.

Martha 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 system greets Martha by saying “Welcome home” and then when she has entered the living 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 at 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 on the event.*

The second example corresponds to an AS that supports the realization of goal named *"Studying AS"*.

Suppose that John is using a number of objects to support the studying activity at his writing desk, according to his profile (his preferable level of light, temperature, etc.). In this case, John has set as a goal to study. This goal can be decomposed in a hierarchy of abstract tasks that constitute a task model for the goal: sit on a chair, move the chair in proximity to the desk; take the book; place the book on top of the desk; turn on the light. In the AmI environment an AS is formed to support the specific goal, by using four artefacts, a lamp, a chair, a desk and a book. The application logic can be stated as follows: when the chair is occupied and it is near the desk and the book is open on the desk, the lamp is turned on (reading activity has been inferred). The implementation of such a task specification can be represented as a graph of connected services provided by the artefacts.

Furthermore, John can move in the room and change his reading spot at the sofa. This causes an adaptation in the configuration of the Studying AS since a new artefact (sofa) is added and one is removed (desk). Another implication of this mobility is that the light service will adapt to the new reading spot. While reading at the desk the desk light is used, and when he moves to the sofa the lamp near the sofa is used. This implies that device selection for instantiating/adapting an AS depends on user location.

Since workflows are essentially graphs of activities, it is useful to express those using UML activity diagrams. 1.13 describes the sequence of activities for the example scenario. Note that the tasks "AdjustLights", "AdjustAC", "ShowPhotos", and "PlayMusic" can run in parallel and therefore they have been enclosed in a fork-join block. Note also that the exception events are not part of the workflow description but they are handled by the corresponding ATRACO active entities.

1.4.2 Late Binding

We have developed a service composition mechanism which includes 3 phases: *task workflow planning*, *dynamic service binding* and *execution management and control* as illustrated in Figure 1.5.

The planning problem can be stated as "discover an execution path of services (tasks) given some state of the world to achieve a goal". In ATRACO, we use a library of abstract plans which model specific user goals. An abstract plan 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 SM during runtime. Having an abstract service workflow description, which is given in a BPELlike language, the Dynamic Service Binding module of the SM applies a

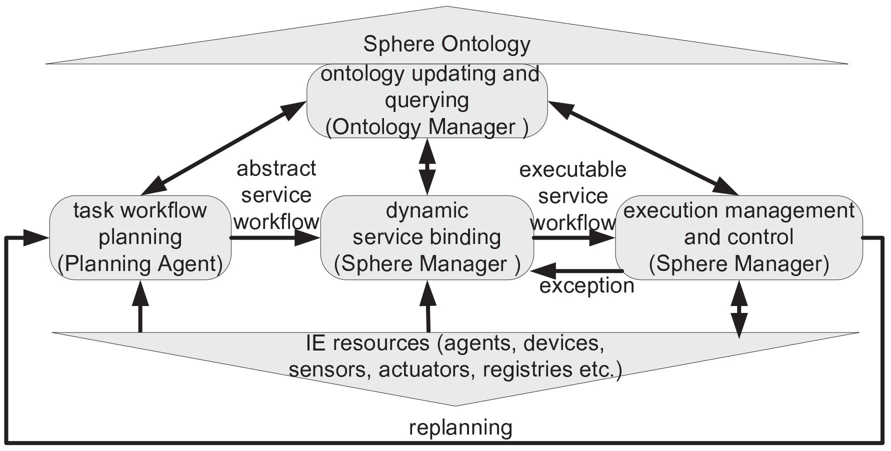
Sphere Ontology
ontology updating and querying (Ontology Manager)
abstract service workflow
executable service workflow
task workflow planning (Planning Agent)
dynamic service binding (Sphere Manager)
execution management and control (Sphere Manager)
exception
IE resources (agents, devices, sensors, actuators, registries etc.)
replanning

Fig. 1.5 Late service composition process in ATRACO.

semanticbased discovery mechanism and uses information about available services and context 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.

This adaptation has been inspired by the subtype polymorphism found in the object-oriented programming paradigm [2]. The concept is that we can adapt the instantiation of the AS to different environments provided that a late binding mechanism is in place that determines the exact resources that will be used in the AS (i.e. the specific artefacts, e.g. “the lamp in the corner”). The different resources that may be involved only need to present a compatible interface to the clients (i.e., in our case, a UPnP interface). Figure 1.6 gives a conceptual view of the dynamic service binding process. A workflow is mapped into a number of tasks and a workflow task is mapped into one or more abstract services. In addition, each service would also require certain physical resources for its implementation. Mapping of the task to the services can be specified at design time by the PA as per users’ functional requirements. However, mapping of the service to the actual human and physical resources is done at runtime, in keeping with service orientation. This dynamic binding is therefore dependent on the context in which the binding occurs.

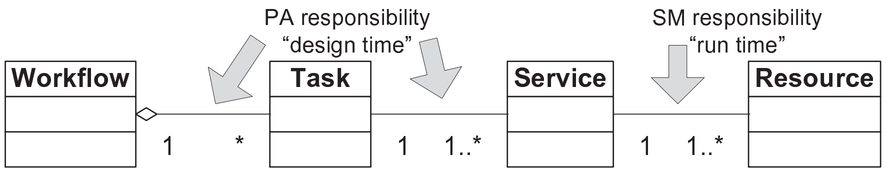

Fig. 1.6 Conceptual model for dynamic service binding.

In the absence of the Sphere Ontology, which has not been yet instantiated, the SM implements a lightweight Resource Discovery Protocol for artefacts or eEntities (eRDP) where the term resource is used as a generalization of the term service. eRDP is a protocol for advertisement and location of network/device resources with a semantic description. The assumption here is that there is a local ontology to describe the services/resources that each artefact can provide and as such assist the service discovery mechanism. In order to support this functionality, an Ontology Manager (OM) is assumed present that provides methods that query this ontology for the services that the artefact provides. The details of the eRDP design and implementation can be found in [17]. The matching resources are returned by eRDP and the SM selects the best set of device(s)/service(s) based on a scoring mechanism that will be explained later. Subsequently, the SM invokes the OM to create the Sphere Ontology (SO) which will include links to all the relative devices to the AS that have been discovered.

After service binding the SM starts any interaction task in conjunction with the IA and also any FTA task and executes the workflow preserving the precedence constraints or the conditions that are specified in the workflow. At runtime 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 on the status of the resource and operates according to the pattern specified by its type. The sequence diagram in Figure 1.7 shows the basic interaction of the software components during the instantiation of the "Feel Comfortable" AS, which employs the dynamic service binding process mentioned earlier. In the diagram, this process is implemented by the methods used inside the two loops.

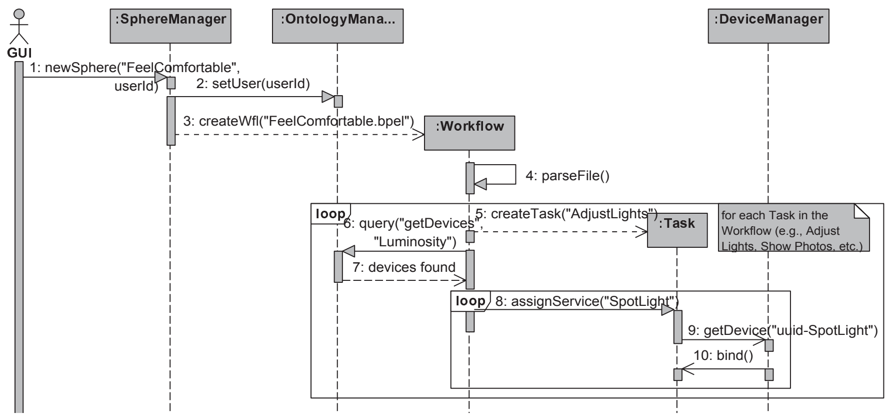
GUI
:SphereManager
:OntologyMana...
:DeviceManager
1: newSphere("FeelComfortable", userId)
2: setUser(userId)
3: createWfl("FeelComfortable.bpel")
:Workflow
4: parseFile()
loop
5: createTask("AdjustLights")
:Task
for each Task in the Workflow (e.g., Adjust Lights, Show Photos, etc.)
6: query("getDevices", "Luminosity")
7: devices found
loop
8: assignService("SpotLight")
9: getDevice("uuid-SpotLight")
10: bind()

Fig. 1.7 Example AS instantiation and binding of devices to services.

The Task object “AdjustLights” is assigned to the FTA component to generate adaptive models for the individual devices/artefacts and for the user behaviours. Figure 1.8 illustrates the initialization of the FTA component to control the room lights in the example AS. The FTA is initialized by passing the input/output, light level related devices as well as light controls which are in turn retrieved from the Sphere Ontology which has been populated with the required ontologies during the AS instantiation. In addition, if the user profile stores initial light preferences (for example from previous executions of the FTA) these can be passed to the FTA in the form of a rule base.

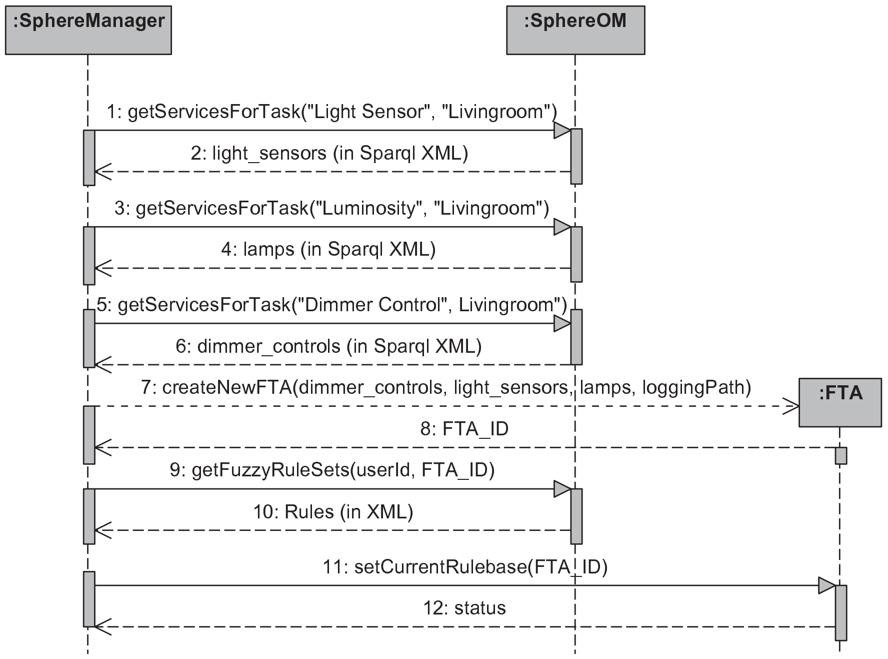
:SphereManager
:SphereOM
:FTA
1: getServicesForTask("Light Sensor", "Livingroom")
2: light_sensors (in Sparql XML)
3: getServicesForTask("Luminosity", "Livingroom")
4: lamps (in Sparql XML)
5: getServicesForTask("Dimmer Control", Livingroom")
6: dimmer_controls (in Sparql XML)
7: createNewFTA(dimmer_controls, light_sensors, lamps, loggingPath)
8: FTA_ID
9: getFuzzyRuleSets(userId, FTA_ID)
10: Rules (in XML)
11: setCurrentRulebase(FTA_ID)
12: status

Fig. 1.8 FTA initialization for the AdjustLights task of the “Feeling Comfortable” AS.

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 preconditions 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. When replanning is requested a new planning problem is defined with the services that are actually available, and the PA solves the problem and delivers a new workflow.

During AS instantiation in IEs there could be multiple devices or services providing similar functionality from which the system will have to choose. Thus, the ATRACO system must provide mechanisms for selection between similar devices or services and decide which of them is the most suitable to participate in the AS. Device selection is based on criteria such as: task suitability, efficiency (as device's proximity to the user, quality of the service or device and stability), user distraction (the inconvenience a user experiences when the system selects different groups of devices than those that the user prefers or is used to use for a specific task) and conflicts with other tasks (more details are given in the Appendix). For calculating the rank for each device we use a scoring mechanism that is similar to that proposed in [29] and is based on multi-attribute utility theory (MAUT). The overall rank of a device given a specific task is defined as a weighted sum of its evaluation with respect to its relevant orthogonal value dimensions (attributes). For ATRACO the relevant value dimensions are scores for task suitability, efficiency, negative of user distraction and negative of conflicts with other tasks.

A ranking policy defines weights between zero and one for each of the above metrics. The scoring policies are defined per task (or task category) by the user and give priority to some of the metrics. E.g., if a task is urgent, the suitability and efficiency ranks must have priority over user distraction, and inter-task conflicts. The weights are normalized to add up to one. The rank of a given device D according to policy P is computed as the dot product of the vector weights specified by the policy with the vector of scores for each one of the metrics. Applying MAUT, the device rank is computed as shown in (1).

$$DR(D,TP) = \sum_{i=1}^{4} w_i(TP) * D(m_i) \tag{1.1}$$

where *DR* is the overall rank of device *D* according to ranking policy *TP* for the task *T*, $w_i(TP)$ is the weight of metric *i* according to policy *TP* and $D(m_i)$ is the rank of device *D* for the metric m_i.

For the task suitability and efficiency we have $D(m) = DS(m)$, while for user distraction and inter-task conflict that have a negative meaning we have $D(m) = 1 - DS(m)$, where $DS(m)$ is the device's score for the metric normalized from 0 to 1.

1.4.3 Ontology Manager

The Ontology Manager (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, the Privacy and Policy Ontology as well as the eventual Sphere Ontology (SO) that emerges from the alignment of all the previous ontologies. The ontology alignment process can be described as: given two ontologies, each describing a set of discrete entities (which can be classes, properties, rules, predicates, or even formulas), find the correspondences, e.g., equivalences or subsumptions, holding between these entities. Thus, under the request of the SM, the OM produces ontology alignments, responds to queries regarding the state or properties of sphere resources, and creates inferences in order to enrich the SO as specified in [38].

The OM has been developed as a wrapper around the Jena Framework (`http://jena.sourceforge.net/`). The OM interface provides comprehensive and simple methods for creating an RDF/OWL based ontology, importing and removing other RDF/OWL based ontologies, updating the ontology at run time, querying of the ontology using SPARQL, and saving the modifications in OWL files. Ontology alignment has been applied by using the Java Alignment API (`http://alignapi.gforge.inria.fr/align.html`). After the alignment, inference and querying is performed on a grid of imported ontologies, given the alignment points that have been produced using OWL class and individual equivalence assertions.

Figure 1.9 illustrates a small sample of the OM interface that is used, for example, to query an ontology using SPARQL syntax, and methods related to the User Profile Ontology e.g., for importing and exporting rules from the FTA.

```
public String queryForSparqlXML(String query, boolean autoPrefix,
String queryType)
```

Performs a query to the ontology. autoPrefix determines if OM will try to resolve known prefixes and the queryType can be ASK or Select. Returns the results in SparqlXML format.

```
public String[] getFuzzyRuleSets(String userId, String FTA_ID) throws
Exception
```

Returns in XML format the stored fuzzy rulesets that match the given userId and FTA_ID. Used by SM to retrieve stored fuzzy rule sets during initialization of the corresponding FTA.

```
public String getServicesForTask(String serviceDescription, String
location) throws Exception
```

Returns a Sparql XML string containing technical parameters for each device and service that matches the serviceDescription description tag and is within the location specified by the second parameter. Used by SM for resolving tasks to specific devices, services, actions, variables and values.

Fig. 1.9 A sample of the OM interface.

A number of ontologies have been developed for and used in the prototype for the representation of AS high level concepts (Figure 1.3), devices and their services, and users and their profile information.

The User Profile ontology holds personal information about the user. It consists of a local, private OWL ontology file that contains the actual user information in the form of individuals and assertions and a publicly accessible (via HTTP) ontology that contains the generic classes, properties and restrictions that describe a user profile. Currently User Profile Ontology contains assertions about the Social Profile (name, nickname, email, address etc.), the location, the activities (Goals and Plans) and the preferences of the user (in the form of stored fuzzy rule sets).

Figure 1.10 illustrates part of an instance of a device ontology for one of the spot lamps used in the prototype. The Service concept represents an abstract service that the device can provide enriched with descriptive tags e.g., a lamp can provide lighting service. In general, a device may offer more than one service and thus more Service instances may be defined. The `StateVariable` concept represents the abstract states of the corresponding Service. It encapsulates the linguistic variable and labels that are required by the FTA for the creation of adaptive device models. The device ontology includes technical characteristics and information about communication with the device in the context of a UPnP environment. Finally, the name, the owner, the location and physical properties of the device are included.

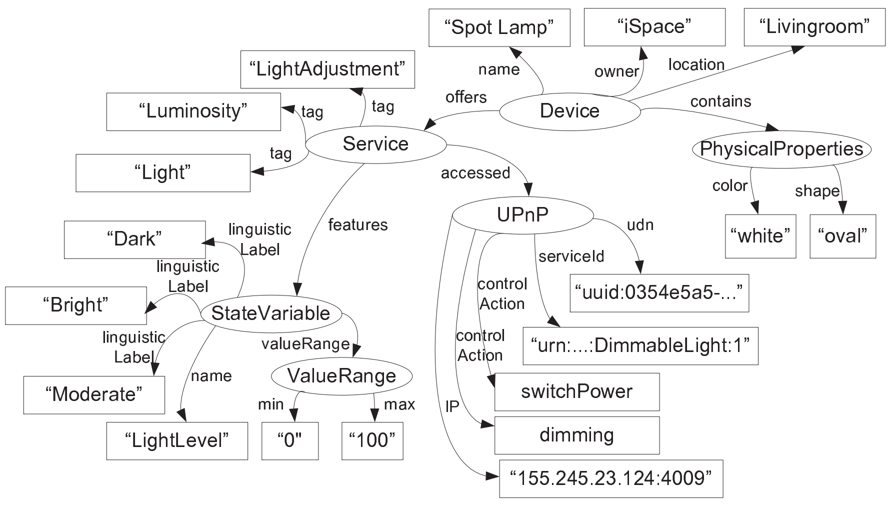

Fig. 1.10 Part of an instance of a device ontology for a spot lamp.

1.4.4 ATRACO-BPEL Workflow Specification

BPEL defines a model and a grammar for describing the behavior of a business process based on interactions between the process and its partners. It allows for creating complex processes by creating and wiring together different activities that can, for example, perform Web services invocations (<invoke>), waiting to be invoked by someone externally (<receive>), generate a response (<reply>), manipulate data (<assign>, throw faults (<throw>), or terminate a process (<exit>). In our case, the business process represents the process model of an AS and the partners can take the form, either of a service of a simple device, or the service of an ATRACO agent.

While BPEL is a suitable language for describing workflows, an ATRACO workflow description presents requirements that cannot be completely covered by BPEL. This is due to the following:

i. BPEL partners (partnerLinks) are bound statically to specific Web services. In the context of ATRACO, however, services are not bound at design time but dynamically during the execution of the workflow. Thus, there is a need to describe services in the workflow by their semantics which mainly define ontological related searching terms (for example, "Luminosity" for a light service).
ii. The limitation of the one-to-one mapping of services between communicating partners, supported by BPEL. On the other side, ATRACO tasks may need to handle two or more services that provide input or output to the task.
iii. BPEL supports a single coordinator that executes the orchestration logic. ATRACO workflows normally are centrally handled by the Sphere Manager which implements the workflow execution engine; however a more distributed scheme can also be followed by sharing parts of the workflow with collaborating agents (e.g., IA and FTA). This collaboration sets some special requirements in the description of the workflow.

Given the above requirements a variant of BPEL, called ATRACO-BPEL, was defined in order to provide those ATRACO specific features needed in order specify workflows. In the following we explain how using the ATRACO-BPEL formalism an example task is bound with the appropriate service(s). The task *AdjustLights* is associated with the parnterLink *AdjustLightsPL* as part of the orchestration logic section:

```
<bpel:invoke
  name="AdjustLights" partnerLink="AdjustLightsPL">
</bpel:invoke>
```

The partnerLink *AdjustLightsPL* has an input role called *ATRACO:lightStatus* and an output role (partnerRole) called *ATRACO:triggerLight*. The *Continues* type denotes that the execution of the activity is to be treated as a task that is running continuously, i.e., the workflow does not wait its termination.

```
<bpel:partnerLink
  name="AdjustLightsPL"
  partnerLinkType="ATRACO:Continuous"
  myRole="ATRACO:lightStatus"
  partnerRole="ATRACO:triggerLight">
</bpel:partnerLink>
```

The input role *ATRACO:lightStatus* denotes the appropriate abstract service that must be bound to fulfill the role (Luminosity) along with any other application specific details that are needed for its operation e.g., the task will be monitored by an ATRACO agent for learning user behavior with respect to light adjustments and all found light devices are to be used.

```
<ATRACO:role
  name="lightStatus" type="input" Agent="yes" IAmode ="none">
  <ATRACO:service semantics="Luminosity" trigger="Low" reset
      ="none" quantity="all" rules="">
  </ATRACO:service>
</ATRACO:role>
```

The corresponding definition for the output role will be:

```
<ATRACO:role
  name="triggerLight" type="output" Agent="yes" IAmode="
      withAgent">
  <ATRACO:service semantics="Actuate Light" trigger="On"
      reset="Off" quantity="all" rules="">
  </ATRACO:service>
</ATRACO:role>
```

In ATRACO-BPEL each partnerLink role is specialized as an ATRACO:role which is a new definition in ATRACO-BPEL. In each ATRACO:role the attributes listed in Table 1.3 are defined.

Table 1.3 ATRACO:role semantics in ATRACO-BPEL.

Attribute	Semantics
name	The name of the role.
type	Denotes the type of the role. Accepted values are **input/output**.
Agent	This attribute defines whether the task is monitored by an ATRACO agent or not. Accepted values are **yes/no**.
IAmode	Specifies the interaction mode with the ATRACO Interaction Agent. Accepted values are: **none** no interaction is needed; **pure** this value is used to indicate that a single interaction with the user through a dialog interface (spoken, tangible or software) needs to be provided either to provide a message or to receive an input for the system from the user in a form of question; **direct** this value is used when the IA needs to create an interface for an output device; **withAgent** this value is used to indicate that there is a need to find proper user inputs for the Agent monitored tasks.

Each ATRACO:role envelopes a set of services that are bound to it. Each role can have more than one abstract service. If the role type is input then the activity waits for all the services to deliver their result before proceeding. If the role type is output then, upon activity completion, all the services enveloped in this role are triggered. For each abstract service specific attributes are defined, providing the necessary support for device discovery and service operation. Table 1.4 summarizes the service-specific attributes in ATRACO-BPEL.

Table 1.4 Service-specific attributes in ATRACO-BPEL.

Attribute	Semantics
semantics	The semantics of the service as a set of keywords – these are used to find the specific device that can be bound to this abstract service.
trigger	**input role**: denotes a linguistic value that triggers the service. **output role**: denotes a linguistic value passed to the service.
reset	The reset state (linguistic value) that the service should apply in the case that the activity cannot be performed.
quantity	A number that defines how many devices providing this service are needed for the specific activity. If the value is "**all**" then all found devices are used.
rules	Any special constraints need to be met for binding the corresponding device(s).
IAdlg	This attribute is associated with the direct or pure interaction modes with IA in order to give it the proper interaction dialog type. Examples of accepted values are: GreetingMessage, LightInstructions, GrantGuestAccess, MusicQuestion, MusicControl, PhotoFrameQuestion, SlideshowControl.

1.5 Deployment

In order to test the AS adaptation mechanisms we have implemented an experimental prototype in the AmInOffice testbed. The AmInOffice is a testbed developed in the premises of Dynamic Ambient Intelligent Systems Research Unit at RACTI (`daisy.cti.gr`) and consists of a variety of sensors deployed in the office environment, a set of smart objects that support office tasks and the appropriate network infrastructure. In order to implement the above scenario we have set up AmInOffice with the following devices:

- An RFID reader near the door of the office to read RFID tags
- Two light sensors each one reading light level in a different spot in the office
- Two ceiling lamps controlling the ambient light
- Two lamps one placed at the desk and one near the sofa
- Speakers connected to the main PC for playing music and producing vocal messages
- A smart chair (eChair) able to sense if someone is sitting on it
- A smart sofa (eSofa) that can sense if someone is sitting on it and at which spot (left or right)
- Smart books (eBooks). Apart from smart readers (eReaders) this includes a typical book instrumented with bending sensors that can sense if the book is opened or closed.
- A smart desk (eDesk) that can sense objects on it and near it.

Figure 1.11 illustrates how the devices have been placed in the AmInOffice.

The ATRACO components that implement the necessary functionality in order to support AS formation and adaptation based on mechanisms discussed in this work are the Sphere Manager (SM) and Ontology Manager (OM). Interaction with other ATRACO components such as Planning Agent and Privacy Manager is assumed

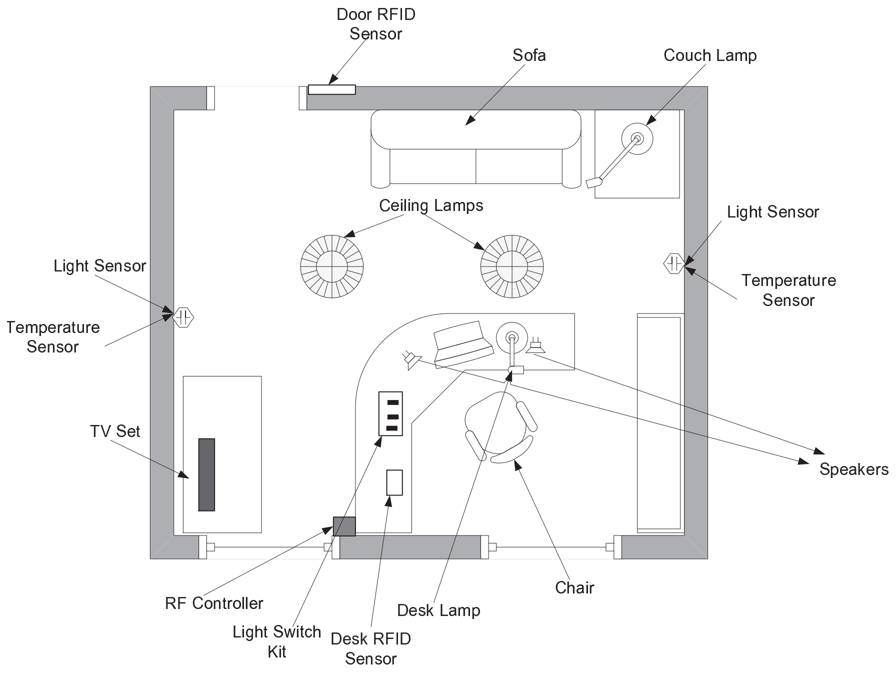

Fig. 1.11 AmInOffice setup for the experimental prototype.

and requires the interfaces specified in [15]. Third-party tools have been also used for performing alignment. The ontologies for all the artefacts used have been developed and a semantically rich UPnP device ontology was developed to support workflow-driven inclusion of UPnP compatible devices in a sphere. In the experimental prototype we have tested the following functionality:

- *Sphere initialization*: Initiate an AS through an ATRACO-BPEL file. Test workflow creation and execution.
- *Late binding of the devices*: Bind abstract services needed for each task to specific devices that exist in AmInOffice in collaboration with the OM at runtime.
- *Runtime application behavior*: Validate that the running tasks correspond to the scenario of the experiment.
- *Handling of adaptation events*: Test system response to adaptation events that affect the configuration of the AS categorized in the following types:
  - *User location change*: test system reaction when the location of the user associated with the AS changes.
  - *Resource not available*: test system reaction when a device bound to a task fails. Check if the system can find an appropriate replacement.
  - *New resource (service/device)*: test system reaction when a new device relevant to the task that is running is available.

– *New person*: test system reaction (in terms of security and privacy) when a new user is recognized by the system

Figure 1.12 illustrates in the form of an activity diagram the main tasks to be executed by the Sphere Manager component in order to handle each one of the above adaptation events. The starting point for running an AS is the generation of the cor-

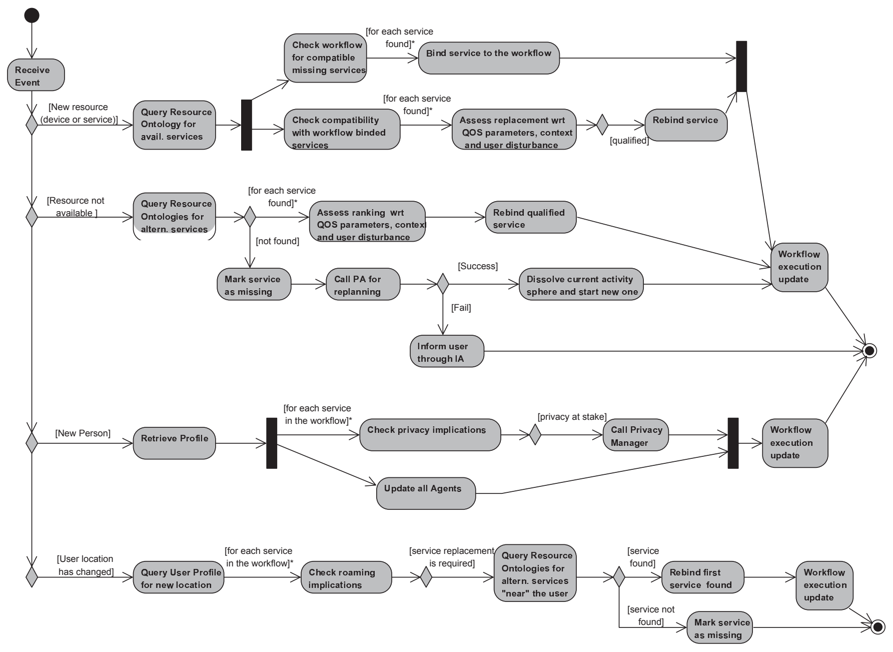
Receive Event
[New resource (device or service)]
Query Resource Ontology for avail. services
Check workflow for compatible missing services
[for each service found]*
Bind service to the workflow
Check compatibility with workflow binded services
[for each service found]*
Assess replacement wrt QOS parameters, context and user disturbance
[qualified]
Rebind service
[Resource not available]
Query Resource Ontologies for altern. services
[for each service found]*
Assess ranking wrt QOS parameters, context and user disturbance
Rebind qualified service
[not found]
Mark service as missing
Call PA for replanning
[Success]
Dissolve current activity sphere and start new one
[Fail]
Inform user through IA
Workflow execution update
[New Person]
Retrieve Profile
[for each service in the workflow]*
Check privacy implications
[privacy at stake]
Call Privacy Manager
Update all Agents
Workflow execution update
[User location has changed]
Query User Profile for new location
[for each service in the workflow]*
Check roaming implications
[service replacement is required]
Query Resource Ontologies for altern. services "near" the user
[service found]
Rebind first service found
[service not found]
Mark service as missing
Workflow execution update

Fig. 1.12 Adaptation events handling.

responding workflow. Workflows are described in ATRACO-BPEL, but they can be represented in a more user friendly way with activity diagrams. The diagram in Fig. 1.13 illustrates the workflow for the “Feel Comfortable” AS of the example scenario. The diagram is annotated with labels from the source file in an attempt to close the gap between the high-level view of the diagram and the low-level view of the file. For example, the annotation in each box shows the activity type in the main sequence and the task name, the ontological searching term, as well as which ATRACO component, besides SM, has responsibility for running parts of this task.

The technical requirements for the deployment and testing of the ATRACO system include: the runtime versions of the ATRACO components with the specified service interfaces; the devices serving the scenarios, wrapped as UPnP devices; the domain and resource ontologies; the workflows specifying the tasks in each AS; and various third-party run-time libraries. The deployment of the system has been done in two IE testbeds using scenarios similar to the one discussed in this paper.

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 [41] is used as the communication middleware for the integration of devices and services, instead of Web services. OWL has been used for the development of the ontologies as it provides a strong logical reasoning framework for the expression and enforcement of ATRACO policies and rules.

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 ATRACO-specific. In addition, most engines do not allow for dynamic binding and discovery of services. 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. This is achieved by encapsulating service search parameters in ATRACO-BPEL (see Table 1.4) as an input to the dynamic service binding process.

When the user changes position in the room the ATRACO system is notified for that change. While this location context can be provided either by using motion detection devices, or specific services such as Ubisense (used in iSpace), for our experiment we emulated such a device by using a WoZ interface and selecting the appropriate location. When the SM receives a location change event, it queries OM for the new location. Then for each service that is bound to the active task it checks if there are any requirements for device replacement. This is done by querying the OM with the new user location context. If the device that OM returns is not equal to the currently bound then it proceeds with service replacement for the appropriate task. The sequence diagram in Figure 1.14 shows the exact messages that are exchanged for the task "Reading" when the user changes location to the sofa.

1.6 Discussion

The SOA approach appears to be a convenient architectural style towards meeting one of the key objectives of the ATRACO project that is the need for adaptable and reconfigurable systems. Analyzing contemporary software technologies complying with the SOA architectural paradigm, such as OSGi, UPnP, and the Web services architecture appears that current software technologies do not meet the adaptability and interoperability requirements for the ATRACO project.

In the first case SOA provides little support on how adaptive services can be used to allow people to interact with an AmI environment in a seamless and unobtrusive manner. In other words, research into service composition has mainly focused on the composition mechanism rather than on guiding composition to enable the user to perform activities in the way they wish to do. A challenge here is how to automate the service composition process, so that the service offered to users appears to be

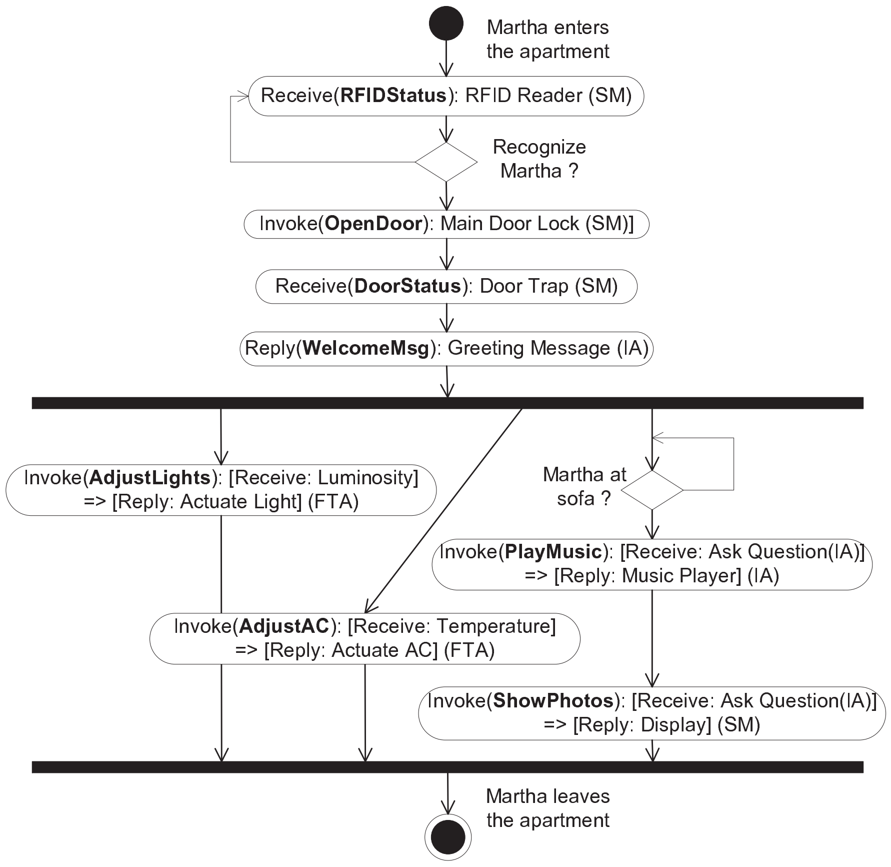
Martha enters the apartment
Receive(**RFIDStatus**): RFID Reader (SM)
Recognize Martha ?
Invoke(**OpenDoor**): Main Door Lock (SM)]
Receive(**DoorStatus**): Door Trap (SM)
Reply(**WelcomeMsg**): Greeting Message (IA)
Invoke(**AdjustLights**): [Receive: Luminosity] => [Reply: Actuate Light] (FTA)
Martha at sofa ?
Invoke(**PlayMusic**): [Receive: Ask Question(IA)] => [Reply: Music Player] (IA)
Invoke(**AdjustAC**): [Receive: Temperature] => [Reply: Actuate AC] (FTA)
Invoke(**ShowPhotos**): [Receive: Ask Question(IA)] => [Reply: Display] (SM)
Martha leaves the apartment

Fig. 1.13 Annotated activity diagram for the "Feel Comfortable" AS.

adaptive, in the sense that the service provided changes dynamically according to the task the user wishes to perform and the context in which they wish to perform it.

In the second case, current solutions provide little support for semantic-based interoperability, hence dealing with interaction between services based on syntactic description for which common understanding is hardly achievable in an open environment. The latter issue may be addressed using semantic modeling through ontologies. Ontologies can provide an extensible and flexible way of expressing the basic terms and their relations in a domain, task or service. However, the issue that can be raised by the heterogeneity of ontologies and how to achieve semantic interoperability between systems using different ontologies remains a challenge. In ATRACO the approach that is followed in order to address this challenge is to research, develop and test theories of ontology alignment to achieve task based semantic integration of heterogeneous devices and services. This issue is examined thoroughly in a separate chapter.

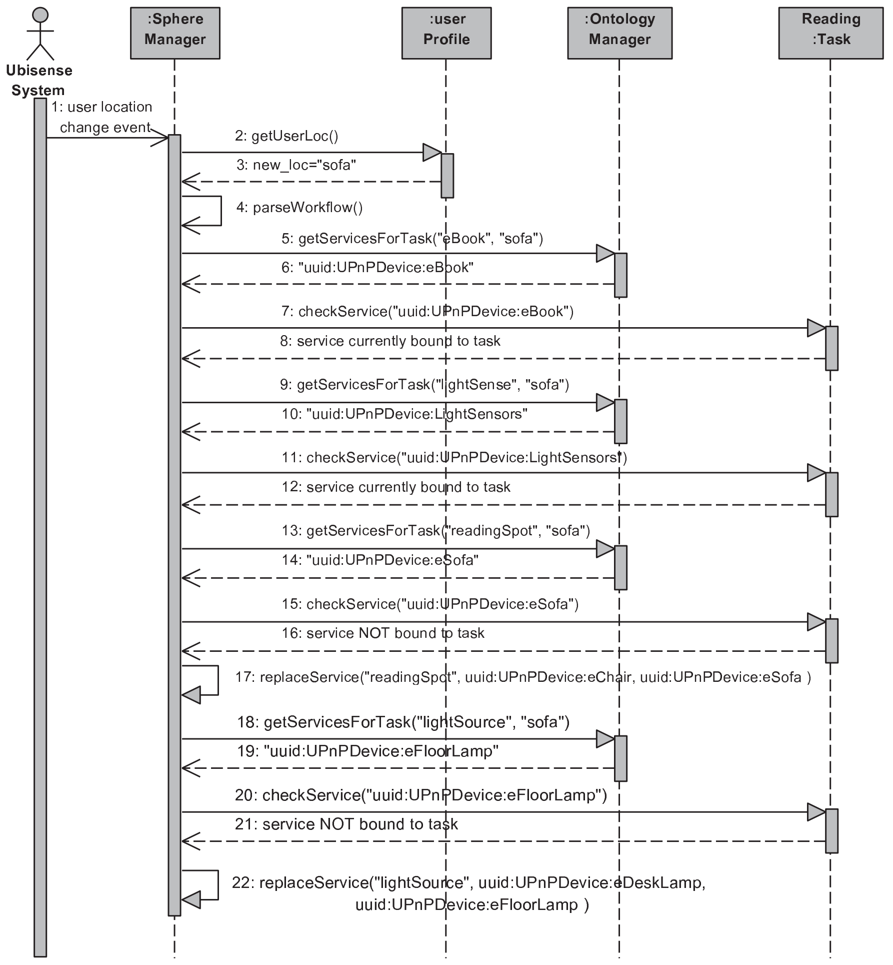
Ubisense System
:Sphere Manager
:user Profile
:Ontology Manager
Reading :Task
1: user location change event
2: getUserLoc()
3: new_loc="sofa"
4: parseWorkflow()
5: getServicesForTask("eBook", "sofa")
6: "uuid:UPnPDevice:eBook"
7: checkService("uuid:UPnPDevice:eBook")
8: service currently bound to task
9: getServicesForTask("lightSense", "sofa")
10: "uuid:UPnPDevice:LightSensors"
11: checkService("uuid:UPnPDevice:LightSensors")
12: service currently bound to task
13: getServicesForTask("readingSpot", "sofa")
14: "uuid:UPnPDevice:eSofa"
15: checkService("uuid:UPnPDevice:eSofa")
16: service NOT bound to task
17: replaceService("readingSpot", uuid:UPnPDevice:eChair, uuid:UPnPDevice:eSofa)
18: getServicesForTask("lightSource", "sofa")
19: "uuid:UPnPDevice:eFloorLamp"
20: checkService("uuid:UPnPDevice:eFloorLamp")
21: service NOT bound to task
22: replaceService("lightSource", uuid:UPnPDevice:eDeskLamp, uuid:UPnPDevice:eFloorLamp)

Fig. 1.14 Replacing the light providing device as a result of user mobility.

ATRACO concrete plans are described as mentioned previously by workflow specifications using an extension of BPEL (ATRACO-BPEL). Normally, workflow management systems have not been used for dynamic environments requiring adaptive behaviour. Typically an intranet-based workflow system executes, by using a collection of services that are owned and managed by the same organization. In this setting, service interruptions are rare and typically they are scheduled during system maintenance. On the contrary, 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. On the other side our approach is viewed as collaborative problem solving approach where a set of autonomous agents work together to achieve a common goal. Our general idea then 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 ATRACO components. Applications can be specified then first with a workflow description using ATRACO-BPEL 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 of the workflow description.

1.7 Conclusion

ATRACO supports the deployment and execution of applications that need to be adapted and reconfigured in dynamic environments. The need for adaptation and reconfiguration calls for a modular design approach, which the SOA paradigm tends to provide. Following this architectural style, each device provides services through which other components can obtain information or control its behaviour. When an application has to be adapted, either during application transition to a new environment or when a device running a service fails, a description of the structure of the application, which is modelled as a workflow of abstract services, is used by an adaptation module which makes use of ontologies, context information and defined policies to generate a new structure for the adapted application. The agent approach complements the SOA modular and flexible infrastructure by providing high level adaptation to user's tasks, as an intelligent control layer above SOA.

Appendix

Table 1.5 Metrics for device selection.

Metric	Description
Task suitability	This quantifies, as a percentage, how well a specific device or service is suitable for a specific task. Its value is calculated based on the semantic relevance between the task's description as it is presented in an abstract plan, and the device's or service's description provided by the device's ontology. E.g., for a service of providing light, both a lamp and a computer monitor can be candidates. The fact that the lamp's ontology states as primary purpose of the device the supply of light while the monitor's ontology state the light emission as a secondary attribute gives the lamp a higher score for this metric.
Efficiency	This metric measures the efficiency of a device or service for a certain goal. It expresses how well or to what degree the device is able to contribute in achieving a goal. Its value is calculated over a combination of other measures such as the device's proximity to the user (based on location info from User Profile Ontology (UPO) and device ontology), the quality of the service or device (as inferred from the specifications of the device that are encoded in its ontology) and the stability that quantifies how well a device will be able to perform a task to completion. The exact measures that participate in the calculation of efficiency are depended on the nature of the task and are derived from the policies encoded in the ATRACO ontologies. E.g., if the goal is to provide enough light for the user to read a book for an hour, a lamp located closer to the user has a strong potential to be selected. But if its light is weaker than the minimum needed for reading and another lamp exists a little more far, but in the range of the user, and can provide the desirable light level, the later should get a higher efficiency score. In a similar way if a device runs out of battery and will not last to achieve fully the goal its efficiency score should be discounted.
User distraction	User distraction expresses the inconvenience a user experiences when the system selects different groups of devices than those that the user prefers or is used to use for a specific task. User's preferences and habits are expected to be stated at the UPO or inferred from it. E.g., if a lamp with strong light is available near the user as he reads his book, but the user has expressed (directly or indirectly) its preference to use a specific one with weaker light when reading at this part of the room, the system should penalize the first lamp by increasing its user distraction score.
Conflicts with other tasks	This quantifies the number of other running tasks that will be blocked by selecting a device. E.g., if a monitor is currently used for watching a film, its conflicting score for selecting it for the task of displaying incoming emails should be greater than zero because it will obstruct the film watching task.

References

1. Alves, A., Arkin, A., Askary, S., Barreto, C., Ben, Curbera, F., Ford, M., Goland, Y., Guízar, A., Kartha, N., Liu, C.K., Khalaf, R., König, D., Marin, M., Mehta, V., Thatte, S., van der Rijn, D., Yendluri, P., Yiu, A.: Web Services Business Process Execution Language Version 2.0. Tech. rep., OASIS Web Services Business Process Execution Language (WSBPEL) TC (2007). URL http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html
2. Armstrong, D.J.: The quarks of object-oriented development. Commun. ACM **49**, 123–128 (2006). DOI http://doi.acm.org/10.1145/1113034.1113040. URL http://doi.acm.org/10.1145/1113034.1113040
3. Bidot, J., Schattenberg, B., Biundo, S.: Intelligent planner. Tech. rep., University of Ulm (2010)
4. Birrell, A.D., Nelson, B.J.: Implementing remote procedure calls. ACM Trans. Comput. Syst. **2**, 39–59 (1984). DOI http://doi.acm.org/10.1145/2080.357392. URL http://doi.acm.org/10.1145/2080.357392
5. Blair, G.S., Coulson, G., Andersen, A., Blair, L., Clarke, M., Costa, F., Duran-Limon, H., Fitzpatrick, T., Johnston, L., Moreira, R., Parlavantzas, N., Saikoski, K.: The design and implementation of open orb 2. IEEE Distributed Systems Online **2**, – (2001). URL http://portal.acm.org/citation.cfm?id=1435643.1436507
6. Brønsted, J., Hansen, K.M., Ingstrup, M.: Service composition issues in pervasive computing. IEEE Pervasive Computing **9**, 62–70 (2010). DOI http://dx.doi.org/10.1109/MPRV.2010.11. URL http://dx.doi.org/10.1109/MPRV.2010.11
7. Capra, L., Blair, G.S., Mascolo, C., Emmerich, W., Grace, P.: Exploiting reflection in mobile computing middleware. SIGMOBILE Mob. Comput. Commun. Rev. **6**, 34–44 (2002). DOI http://doi.acm.org/10.1145/643550.643553. URL http://doi.acm.org/10.1145/643550.643553
8. Chakraborty, D., Joshi, A.: Dynamic service composition: State-of-the-Art and research directions. Tech. rep., University of Maryland, Department of Computer Science and Electrical Engineering (2001)
9. Maciel da Costa, C., da Silva Strzykalski, M., Bernard, G.: An aspect oriented middleware architecture for adaptive mobile computing applications. In: Proceedings of the 31st Annual International Computer Software and Applications Conference - Volume 02, COMPSAC '07, pp. 81–86. IEEE Computer Society, Washington, DC, USA (2007). DOI http://dx.doi.org/10.1109/COMPSAC.2007.59. URL http://dx.doi.org/10.1109/COMPSAC.2007.59
10. Dustdar, S., Schreiner, W.: A survey on web services composition. Int. J. Web Grid Serv. **1**, 1–30 (2005). DOI 10.1504/IJWGS.2005.007545. URL http://portal.acm.org/citation.cfm?id=1358537.1358538
11. Emmerich, W.: OMG/CORBA: An Object-Oriented Middleware. In: J.J. Marciniak (ed.) Encyclopedia of Software Engineering, pp. 902–907. John Wiley & Sons (2002). URL http://www.cs.ucl.ac.uk/staff/w.emmerich/publications/Encyclopedia
12. Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, Upper Saddle River, NJ, USA (2005)
13. Frei, A., Popovici, A., Alonso, G.: Eventizing applications in an adaptive middleware platform. IEEE Distributed Systems Online **6**, 1– (2005). DOI 10.1109/MDSO.2005.20. URL http://portal.acm.org/citation.cfm?id=1069591.1069686
14. Gjørven, E., Eliassen, F., Lund, K., Eide, V.S.W., Staehli, R.: Self-adaptive systems: A middleware managed approach. In: SelfMan, pp. 15–27 (2006)
15. Goumopoulos, C., Calemis, I., Togias, K., Kameas, A., Pruvost, G., Wagner, C., Meliones, A., Wiedersheim, B., Bidot, J.: Integrated component platform for prototype testing and updated specification and design report. Tech. rep., Computer Technology Institute, ATRACO ICT 1.8.2 216837 D7 (2010)

16. Goumopoulos, C., Kameas, A.: Ambient ecologies in smart homes. Comput. J. **52**, 922–937 (2009). DOI http://dx.doi.org/10.1093/comjnl/bxn042. URL `http://dx.doi.org/10.1093/comjnl/bxn042`
17. Goumopoulos, C., Kameas, A.: Smart objects as components of ubicomp applications. International Journal of Multimedia and Ubiquitous Engineering, Special Issue on Smart Object Systems **4**(3), 1–20 (2009). URL `http://www.sersc.org/journals/IJMUE/vol4_no3_2009/1.pdf`. SERSC Press
18. Goumopoulos, C., Kameas, A., Hagras, H., Callaghan, V., Gardner, M., Minker, W., Weber, M., Bellik, Y., Meliones, A.: Atraco: Adaptive and trusted ambient ecologies. In: Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, pp. 96–101. IEEE Computer Society, Washington, DC, USA (2008). DOI 10.1109/SASOW.2008.13. URL `http://portal.acm.org/citation.cfm?id=1524875.1525041`
19. Grace, P., Truyen, E., Lagaisse, B., Joosen, W.: The case for aspect-oriented reflective middleware. In: Proceedings of the 6th international workshop on Adaptive and reflective middleware: held at the ACM/IFIP/USENIX International Middleware Conference, ARM '07, pp. 2:1–2:6. ACM, New York, NY, USA (2007). DOI http://doi.acm.org/10.1145/1376780.1376782. URL `http://doi.acm.org/10.1145/1376780.1376782`
20. Grimes, R.: Professional Dcom Programming. Wrox Press Ltd., Birmingham, UK, UK (1997)
21. Gruber, T.R.: Toward principles for the design of ontologies used for knowledge sharing. Int. J. Hum.-Comput. Stud. **43**, 907–928 (1995). DOI 10.1006/ijhc.1995.1081. URL `http://portal.acm.org/citation.cfm?id=219666.219701`
22. Hayton, R.: Flexinet open orb framework. Tech. rep., APM Ltd., Poseidon House, Castle Park, Cambridge, UK (1997)
23. Hollingsworth, D.: Workflow management coalition - the workflow reference model. Tech. rep., Workflow Management Coalition (1995)
24. Kiczales, G.: Aspect-oriented programming. ACM Comput. Surv. **28** (1996). DOI http://doi.acm.org/10.1145/242224.242420. URL `http://doi.acm.org/10.1145/242224.242420`
25. Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W.G.: An overview of aspectj. In: Proceedings of the 15th European Conference on Object-Oriented Programming, pp. 327–353. Springer-Verlag, London, UK (2001). URL `http://portal.acm.org/citation.cfm?id=646158.680006`
26. Kiczales, G., Rivieres, J.D.: The Art of the Metaobject Protocol. MIT Press, Cambridge, MA, USA (1991)
27. Kon, F., Costa, F., Blair, G., Campbell, R.H.: The case for reflective middleware. Commun. ACM **45**, 33–38 (2002). DOI http://doi.acm.org/10.1145/508448.508470. URL `http://doi.acm.org/10.1145/508448.508470`
28. Kon, F., Román, M., Liu, P., Mao, J., Yamane, T., Magalhã, C., Campbell, R.H.: Monitoring, security, and dynamic configuration with the dynamictao reflective orb. In: IFIP/ACM International Conference on Distributed systems platforms, Middleware '00, pp. 121–143. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2000). URL `http://portal.acm.org/citation.cfm?id=338283.338355`
29. Kumar, R., Poladian, V., Greenberg, I., Messer, A., Milojicic, D.: Selecting devices for aggregation. In: WMCSA, pp. 150–159. IEEE Computer Society, Los Alamitos, CA, USA (2003). DOI http://doi.ieeecomputersociety.org/10.1109/MCSA.2003.1240776
30. Ledoux, T.: Opencorba: A reflective open broker. In: Proceedings of the Second International Conference on Meta-Level Architectures and Reflection, Reflection '99, pp. 197–214. Springer-Verlag, London, UK (1999). URL `http://portal.acm.org/citation.cfm?id=646930.710404`
31. Maes, P.: Concepts and experiments in computational reflection. SIGPLAN Not. **22**, 147–155 (1987). DOI http://doi.acm.org/10.1145/38807.38821. URL `http://doi.acm.org/10.1145/38807.38821`

32. McIlraith, S.A., Son, T.C., Zeng, H.: Semantic web services. IEEE Intelligent Systems **16**, 46–53 (2001). DOI http://dx.doi.org/10.1109/5254.920599. URL `http://dx.doi.org/10.1109/5254.920599`
33. Papadopoulos, N., Meliones, A., Economou, D., Karras, I., Liverezas, I.: A connected home platform and development framework for smart home control applications. In: Proceedings of the 7th IEEE International Conference on Industrial Informatics (INDIN09) (2009)
34. Pawlak, R., Seinturier, L., Duchien, L., Florin, G.: Jac: A flexible solution for aspect-oriented programming in java. In: Proceedings of the Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns, pp. 1–24. Springer-Verlag, London, UK (2001). URL `http://portal.acm.org/citation.cfm?id=646931.710426`
35. Pruvost, G., Kameas, A., Heinroth, T., Seremeti, L., Minker, W.: Combining agents and ontologies to support Task-Centred interoperability in ambient intelligent environments. In: Proceedings of the 2009 Ninth International Conference on Intelligent Systems Design and Applications, ISDA '09, pp. 55–60. IEEE Computer Society, Washington, DC, USA (2009). DOI http://dx.doi.org/10.1109/ISDA.2009.195. URL `http://dx.doi.org/10.1109/ISDA.2009.195`
36. Rao, J., Su, X.: A survey of automated web service composition methods. In: J. Cardoso, A. Sheth (eds.) Semantic Web Services and Web Process Composition, *Lecture Notes in Computer Science*, vol. 3387, pp. 43–54. Springer Berlin / Heidelberg (2005). URL `http://dx.doi.org/10.1007/978-3-540-30581-1_5`
37. Remagnino P Foresti, G.L.: Ambient intelligence: A new multidisciplinary paradigm. IEEE Transactions on Systems, Man and Cybernetics, Part A **35**(1), 1–6 (2005)
38. Seremeti, L., Goumopoulos, C., Kameas, A.: Ontology-based modeling of dynamic ubiquitous computing applications as evolving activity spheres. Pervasive Mob. Comput. **5**, 574–591 (2009). DOI 10.1016/j.pmcj.2009.05.002. URL `http://portal.acm.org/citation.cfm?id=1630161.1630223`
39. Sommer, R.: Personal Space: The Behavioral Basis of Design. Prentice Hall Trade, Englewood Cliffs, NJ, USA (1969)
40. Sycara, K., Paolucci, M., Ankolekar, A., Srinivasan, N.: Automated discovery, interaction and composition of semantic web services. Journal of Web Semantics **1**(1), 27–46 (2003)
41. Togias, K., Goumopoulos, C., Kameas, A.: Ontology-Based representation of upnp devices and services for dynamic Context-Aware ubiquitous computing applications. In: International Conference on Communication Theory, Reliability, and Quality of Service, pp. 220–225. IEEE Computer Society, Los Alamitos, CA, USA (2010). DOI http://doi.ieeecomputersociety.org/10.1109/CTRQ.2010.44
42. Verma, K., Sivashanmugam, K., Sheth, A., Patil, A., Oundhakar, S., Miller, J.: Meteor-s wsdi: A scalable p2p infrastructure of registries for semantic publication and discovery of web services. Inf. Technol. and Management **6**, 17–39 (2005). DOI 10.1007/s10799-004-7773-4. URL `http://portal.acm.org/citation.cfm?id=1047575.1047628`
43. Wagner, C., Hagras, H.: Toward general type-2 fuzzy logic systems based on zslices. Trans. Fuz Sys. **18**, 637–660 (2010). DOI http://dx.doi.org/10.1109/TFUZZ.2010.2045386. URL `http://dx.doi.org/10.1109/TFUZZ.2010.2045386`
44. Wollrath, A., Riggs, R., Waldo, J.: A distributed object model for the java system. Computing Systems **9**(4), 265–290 (1996)
45. Yang, Z., Cheng, B.H.C., Stirewalt, R.E.K., Sowell, J., Sadjadi, S.M., McKinley, P.K.: An aspect-oriented approach to dynamic adaptation. In: Proceedings of the first workshop on Self-healing systems, WOSS '02, pp. 85–92. ACM, New York, NY, USA (2002). DOI http://doi.acm.org/10.1145/582128.582144. URL `http://doi.acm.org/10.1145/582128.582144`