Advances in Ubiquitous Computing:
Future Paradigms and Directions

Activity-oriented Computing

João Pedro Sousa
Computer Science Dept., George Mason University, Fairfax VA
ph: 703-993-9196, fax: 703-993-1638, email: jpsousa@gmu.edu

Bradley Schmerl
Computer Science Dept., Carnegie Mellon University, Pittsburgh PA
ph: 412-268-5889, fax: 412-268-5576, email: schmerl@cs.cmu.edu

Peter Steenkiste
Computer Science Dept., Carnegie Mellon University, Pittsburgh PA
ph: 412-268-3261, fax: 412-268-5576, email: prs@cs.cmu.edu

David Garlan
Computer Science Dept., Carnegie Mellon University, Pittsburgh PA
ph: 412-268-5056, fax: 412-268-5576, email: garlan@cs.cmu.edu

This material is based upon work supported by
the National Science Foundation under Grant No. 0615305

Activity-oriented Computing

ABSTRACT

This chapter introduces a new way of thinking about software systems for supporting the activities of end-users. In this approach, models of user activities are promoted to first class entities, and software systems are assembled and configured dynamically based on activity models. This constitutes a fundamental change of perspective over traditional applications: activities take the main stage and may be long-lived, whereas the agents that carry them out are plentiful and interchangeable.

The core of the chapter describes a closed-loop control design that enables activity-oriented systems to become self-aware and self-configurable, and to adapt to dynamic changes both in the requirements of user activities and in the environment resources. The chapter discusses how that design addresses challenges such as user mobility, resolving conflicts in accessing scarce resources, and robustness in the broad sense of responding adequately to user expectations, even in unpredictable situations, such as random failures, erroneous user input, and continuously changing resources.

The chapter further summarizes challenges and ongoing work related to managing activities where humans and automated agents collaborate, human-computer interactions for managing activities, and privacy and security aspects.

KEYWORDS

Agent, Agent Software, Applications Software, Activity Oriented Computing, Adaptive Systems, Computer Support Systems, Conceptual Model, Emerging Information technologies, End-User Computing, End-User Programming, Healthcare Systems, Home Computing, Human Computer Systems, Human/computer interaction, Middleware, Mobile Computing, Modeling Languages, Office Systems, Office Automation, Personal Computing, Person/machine interaction, Pervasive Computing, Self-* Systems, Software Architecture, Task oriented Computing, Technology Trends, User Interface, Very High-Level Languages, Ubiquitous Computing, Utility Function Evaluation.

INTRODUCTION

Over the past few years, considerable effort has been put into developing networking and middleware infrastructures for ubiquitous computing, as well as in novel human-computer interfaces based on speech, vision, and gesture. These efforts tackle ubiquitous computing from two different perspectives, systems research and HCI research, hoping to converge and result in software that can support a rich variety of successful ubiquitous computing applications. However, although examples of successful applications exist, a good understanding of frameworks for designing ubiquitous computing applications is still largely missing.

A key reason for the lack of a broadly applicable framework is that many research efforts are based on an obsolete application model. This model assumes that ubiquitous computing applications can support user activities by packaging, at design time, a set of related functionalities within a specific domain, such as spatial navigation, finding information on the web, or online chatting. However, user **activities** may require much diverse functionality, often spanning different domains. Which functionalities are required to support an activity can only be determined at runtime, depending on the user needs, and may need to evolve in response to changes in those needs. Examples of user activities targeted by ubiquitous computing are: navigating spaces such as museums, assisting debilitated people in their daily living, activities at the office such as producing reports, as well as activities in the home such as watching a TV show, answering the doorbell, or enhancing house security.

This chapter introduces activity-oriented computing (AoC) as a basis for developing more comprehensive and dynamic applications for ubiquitous computing. Activity-oriented computing brings user activities to the foreground by promoting models of such activities to first class primitives in computing systems.

In the remainder of this chapter, the section on background presents our vision for activity-oriented computing and compares it with related work. Next we discuss the main challenges of this approach to ubiquitous computing. Specifically, we discuss user mobility (as opposed to mobile computing), conflict resolution and robustness, mixed-initiative control, human-computer interaction, and security and privacy.

The main body of the chapter presents our work towards a solution. Specifically we discuss software architectures for activity-oriented computing and how to address the challenges of mobility and robustness, as well as the options to model user activities. The chapter ends with a discussion of future directions concerning human-computer interactions, and the tradeoff between ubiquity and security and privacy.

BACKGROUND

The vision of AoC is to make the computing environment aware of *user **activities*** so that resources can be autonomously managed to optimally assist the user. Activities are everyday actions that users wish to accomplish and that may be assisted in various ways by computing resources in the environment. Done right, AoC will allow users to focus on pursuing their activities rather than on configuring and managing the computing environment. For example, an AoC system could reduce overhead by automatically

customizing the environment each time the user wishes to resume a previously interrupted long-lived activity, such as preparing a monthly report, or organizing a party.

To help make this vision concrete, the following examples illustrate possible applications of AoC.

Elderly care. Rather than relying on hardcoded solutions, AoC enables domain experts such as doctors and nurses to "write prescriptions" for the activities of monitoring the health of the elderly or outpatients. Such descriptions enable smart homes to take charge of those activities, collaborating with humans as appropriate. For example, the heart rate of an elderly person may be monitored by a smart home, which takes responsibility to alert family members when alarming measurements are detected. Who gets alerted and the media to convey the alert may depend on contextual rules, such as the seriousness of the situation, as prescribed by the doctor; the elder's preferences of who to contact, who is available, who is closer to the elder's home, is sending an SMS appropriate, etc.

Entertainment. While others have explored the vision that music, radio, or television can follow occupants as they move through the house, activity-oriented computing enables a more general approach. Entertainment can be defined as an activity, allowing preferences and constraints to be specified, and underlying **services** to be shared, e.g., tracking people, identifying and using devices in various rooms. For example, the same location services used for home security activities can be used for entertainment; the television that can be used for entertainment can also be used for displaying images of a visitor at the front door.

Home Security. Many homes have a security system that uses sensors to detect burglary attempts and fires. They are standalone systems with limited capabilities, e.g., the system is typically either on or off and control is entirely based on a secret code. If the security system were built as an activity service, it could be an open system with richer functionality. For example:

- Richer set of control options, e.g., based on fingerprint readers or voice recognition. These methods may be more appropriate for children or the elderly.
- More flexibility (e.g., giving neighbors limited access to water the plants when the homeowners are on vacation, the ability to control and interact with the system remotely, or incorporate cameras that ignore dogs).
- Remote diagnosis, e.g., in response to an alarm, police or fire responders may be able to quickly check for false alarms through cameras.

Doorbell. A very simple activity is responding to somebody ringing the doorbell. Today's solution is broadcast: the doorbell is loud enough to alert everybody in the house and then people decide independently or after coordination (through shouting!) how to respond. In activity-oriented computing, a doorbell activity carried out by the hallway selects a person, based on their current location, current activity, and age. If the visitor can be identified, it might be possible to have the person who is being visited respond. Also, the method of alerting the person can be customized, e.g., using a (local) sound, displaying a message on the television screen, or flashing the lights. Finally, if nobody is home, the doorbell service can take a voice message or, if needed, establish a voice or video link to a house occupant who might be available in their office or car. Activities such as answering the phone could be handled in a similar way, i.e., replace the broadcast ringing by a targeted, context-aware alert.

What is Activity-oriented Computing

Activity-oriented computing adopts a fundamental change of perspective over traditional applications: **activities** take the main stage and may be long-lived, whereas the agents that carry them out are plentiful and interchangeable; how activities are best supported will evolve over time, depending on the user's needs and context. In AoC, activities are explicitly represented and manipulated by the computing infrastructure. Broadly speaking, this has two significant advantages.

First, it enables explicit reasoning about user **activities**: which activities a user may want to carry out in a particular context, what functionality (**services**) is required to support an activity, what are the user preferences relative to quality of service for each different activity, which activities conflict, which have specific privacy or security concerns, etc.

Second, it enables reasoning about the optimal way of supporting activities, through the dynamic selection of services (agents) that implement specific functions relevant to the activity. Thanks to the explicit modeling of the requirements of activities and of the capabilities of agents, the optimality of such assignment may be addressed by quantitative frameworks such as **utility** theory. Also, by raising the level of abstraction above particular applications or implementations, activity models make it easier to target a broad range of concrete implementations of similar services in different devices, in contrast to solutions based on mobile code (more in the Challenges section, below).

Related Work

Early work in ubiquitous computing focused on making certain applications ubiquitously available. For that, it explored OS-level support that included location sensing components to automatically transfer user interfaces to the nearest display. Examples of this are the work on teleporting X Windows desktops (Richardson, Bennet, Mapp, & Hopper, 1994); and Microsoft's Easy Living project (Brumitt, Meyers, Krumm, Kern, & Shafer, 2000). This idea was coupled with the idea of desktop management to treat users' tasks as sets of applications independent of a particular device. Examples of systems that exploit this idea are the Kimura project (MacIntyre et al., 2001), which migrates collections of applications across displays within a smart room, and earlier work in Aura that targets migration of user tasks across machines at different locations (Wang & Garlan, 2000). Internet Suspend-Resume (ISR) requires minimal changes to the operating system to migrate the entire virtual memory of one machine to another machine (Kozuch & Satyanarayanan, 2002). These approaches focus on making applications available ubiquitously, but do not have a notion of user activity that encompasses user needs and preferences, and therefore do not scale to environments with heterogeneous machines and varying levels of service.

More recent work seeks to support cooperative tasks in office-like domains, for example ICrafter (Ponnekanti, Lee, Fox, & Hanrahan, 2001) and Gaia (Román et al., 2002); as well as domain-specific tasks, such as healthcare (Christensen & Bardram, 2002) and biology experiments, for example, Labscape (Arnstein, Sigurdsson, & Franza, 2001). This research shares with ours the goal of supporting activities for mobile users, where activities may involve several services in the environment, and environments may contain heterogeneous devices. However, much of this work is predicated on rebuilding,

or significantly extending, operating systems and applications to work over custom-built infrastructures. The work described in this chapter supports user activities with a new software layer on top of existing operating systems and accommodates integration of legacy applications.

Focusing on being able to suspend and resume existing activities in a ubiquitous environment does not go all the way toward the vision of providing ubiquitous assistance for user activities. Such support can be divided into two categories: 1) helping to guide users in conducting tasks; and 2) performing tasks, or parts of tasks, on behalf of users. An early example of the first category the Adtranz system (Siewiorek, 1998), which guides technical staff through diagnosing problems in a train system. More recent work concentrates on daily life, often for people with special needs, such as the elderly, or those with debilitated health (Abowd, Bobick, Essa, Mynatt, & Rogers, 2002; Intille, 2002).

Research on automated agents took assistance one step further by enabling systems to carry out activities on behalf of users. Examples of this are the RETSINA framework (Sycara, Paolucci, Velsen, & Giampapa, 2003), with applications in domains such as financial portfolio management, ecommerce and military logistics; and more recently the RADAR project (Garlan & Schmerl, 2007), which focuses on the office domain, automating such tasks as processing email, scheduling meetings, and updating websites.

Consumer solutions for activities in the home are beginning to emerge, mainly from the increasing complexity of configuring home theater equipment. Universal remote controls, such as those provided by Logitech, allow users to define activities such as "Watch DVD", which choose the input source for the television, output of sound through the home theater system, and choosing the configuration of the DVD player (Logitech). However, in these solutions, activities are bound to particular device and device configurations – the activities themselves must be redefined for different equipment, and it is not possible for the activities to move around different rooms in the home, or to allow different levels of service for the same activity.

In this chapter, we discuss the potential and the challenges of having software systems using activity models at runtime. Specifically, we focus on the benefits of using activity models for enabling users to access their activities ubiquitously, and for delegating responsibility for activities to automated agents.

CHALLENGES

Activity-oriented computing raises a number of challenges that must be addressed by any adequate supporting infrastructure and architecture.

User mobility: As users move from one environment to another – for example, between rooms in a house – activities may need to migrate with the users, tracking their location and adapting themselves to the local situation. A key distinction between user mobility in AoC and previous approaches is that no assumptions are made with respect to the users having to carry mobile devices, or to the availability of a particular kind of platform at every location. Since different environments may have very different resources (devices, services, interfaces, etc.) a critical issue is how best to retarget an activity to a new situation. For example, an activity that involves "watching a TV show" can be changed into "listening" when the user walks through a room that only offers audio support.

Solving this problem requires the ability to take advantage of context information (location, resource availability, user state, etc.) as well as knowledge of the activity requirements (which services are required, fidelity tradeoffs, etc.) to provide an optimal use of the environment in support of the activity.

Conflict resolution: Complicating the problem of automated configuration and reconfiguration is the need to support multiple activities – both for a single user and between multiple users. If an individual wants to carry out two activities concurrently that may need to use shared resources, how should these activities simultaneously be supported? For example, if the user is engaged in entertainment, should the doorbell activity interrupt that activity? Similar problems exist when two or more people share an environment. For example, if two users enter the living room hoping to be entertained, but having different ideas of what kind of entertainment they want, how can those conflicts be reconciled? Solving this problem requires (a) the ability to detect when there may be conflicts, and (b) the ability to apply conflict resolution policies, which may itself require user interaction.

Mixed-initiative control: The ability to accomplish certain kinds of activities requires the active participation of users. For example, the door answering activity, which might be associated with a house, requires occupants of the house to respond to requests from the house to greet a visitor. Since humans exhibit considerable more autonomy and unpredictability than computational elements, it is not clear how one should write the activity control policies and mechanisms to allow for this. Standard solutions to human-based activities (such as work-flow management systems) are likely not to be effective, since they assume users to adhere to predetermined plans to a much higher degree than is typically the case in the kinds of domains targeted by ubiquitous computing.

Security and privacy: Some security and privacy issues can be solved through traditional mechanisms for security, but others are complicated by key features of ubiquity: rich context information, and user mobility across public or shared spaces such as a car, or an airport lounge. In a multi-user environment with rich sources of context information (such as a person's location) an important issue is how to permit appropriate access to and sharing of that information. Furthermore, which guaranties can be made to a user that wishes to access personal activities in a shared space? What mechanisms can back such guaranties? Are there deeper issues than the exposure of the information that is accessed in a public space? Is it possible that all of a user's information and identity may be compromised as a consequence of a seemingly innocuous access at a public space?

Human-computer interaction: Many of the activities that a ubiquitous computing environment should support will take place outside of standard computing environment (such as a networked office environment). In such environments one cannot assume that users will have access to standard displays, keyboards, and pointing devices. How then can the system communicate and interact with users effectively? What should be the role of emerging technologies such as augmented reality and natural interaction modalities such as speech, gesture, and ambient properties such as light and smell?

While the challenges above stem from the problem domain, we now turn to the challenges associated with building a solution for AoC.

Activity models. The first challenge is to define what kinds of knowledge should be imparted in systems to make them aware of user activities. Specifically, what should be the contents and form of activity models? What should be the semantic primitives to compose and decompose activities? At what level of sophistication should activity models be captured? Presumably, the more sophisticated the models, the more a system can do to assist users. For example, to help users with repairing an airplane or with planning a conference, a significant amount of domain knowledge needs to be captured. But obviously, capturing such knowledge demands more from users (or domain experts) than capturing simple models of activities. Is there an optimal level of sophistication to capture activity models – a sweet spot that maximizes the ratio between the benefits of imparting knowledge to systems and the costs of eliciting such knowledge from users? Or is it possible to have flexible solutions that allow incremental levels of sophistication for representing each activity, depending on the expected benefits and on the user's willingness to train the system?

System design. Systems that support AoC should be capable of dynamic reconfiguration in response to changes in the needs of user activities. Ideally, such systems would also be aware of the availability of resources in the environment and respond to changes in those. The questions then become: What is an appropriate architecture to support activity-oriented computing? What responsibilities should be taken by a common infrastructure (middleware) and which should be activity- or service-specific? What are the relevant parameters to guide service discovery (location, quality of service, etc.) and how should discovery be geographically scoped and coordinated? Can activity models be capitalized to handle the heterogeneity of the environment, self-awareness and dynamic adaptation? Furthermore, what operations might be used to manage activities: suspend and resume, delegate, collaborate, others? What should be the operational semantics of each of these operations?

Robustness. In AoC, robustness is taken in the broad sense of responding adequately to user expectations, even in unpredictable situations, such as random failures, erroneous user input, and continuously changing resources. First of all, should adequacy be a Boolean variable – either the system is adequate or it is not – or can it be quantified and measured? Specifically, are there system capabilities and configurations that are more adequate then others to support a user's activity? If so, can measures of adequacy be used to choose among alternatives in rich environments? For example is the user better served by carrying out a video conference on a PDA over a wireless link, or on the wall display down the hall?

TOWARDS A SOLUTION

To address the challenges identified above, we decided to start with relatively simple models of activities and address concrete problems where the advantages of AoC could be demonstrated. This section summarizes our experience of about six years at Carnegie Mellon University's Project **Aura**. Initially, this research targeted the smart office domain, and later extended to the smart home domain (more below).

Designing systems for AoC brings up some hard questions. What makes those questions especially challenging, is that to answer them, we need to reexamine a significant number of assumptions that have been made about software for decades. Not

surprisingly, our own understanding of how to answer those questions continues to evolve. This section is organized around the set of solution-related challenges identified above; namely, system design, activity models, and robustness.

System Design

The first research problem we focused on, starting around the year 2000, was user mobility in the smart office domain. Here, activities (or tasks) typically involve several applications and information assets. For instance, for preparing a presentation, a user may edit slides, refer to a couple of papers on the topic, check previous related presentations, and browse the web for new developments. An example of user mobility is that the user may start working on the presentation at his or her office, continue at the office of a collaborator, and pick the task up later at home.

The premise adopted for user mobility is that users should not have to carry a machine around, just as people normally don't carry their own chairs everywhere. If they so desire, users should be able to resume their tasks, on demand, with whatever computing systems are available. This premise is neither incompatible with users carrying mobile devices, nor with mobile code. Ideally, the capabilities of any devices or code that travel with the user contribute to the richness of the environment surrounding the user, and therefore contribute to a better user experience. A discussion of why solutions centered on mobile devices, mobile code, or remote computing (such as PC Anywhere) are not entirely satisfactory to address user mobility can be found in (J.P. Sousa, 2005).

Designing a solution to support user mobility is made harder by the heterogeneity of devices where users may want to resume their activities, and by dynamic variations in the resources and devices available to the user. Even in a fairly restricted office domain, it is common to find different operating systems, offering different suites of applications (e.g. Linux vs. PC vs. Mac.) In a broader context, users may want to carry over their activities to devices with a wide range of capabilities, from handhelds to smart rooms. In addition to heterogeneity, mobile devices are subject to wide variations of resources, such as battery and bandwidth. Ideally, software would automatically manage alternative

Table 1 Terminology

task	An everyday activity such as preparing a presentation or writing a report. Carrying out a task may require obtaining several *services* from an *environment*, as well as accessing several *materials*.
environment	The set of *suppliers*, *materials* and *resources* accessible to a user at a particular location.
service	Either (a) a service type, such as printing, or (b) the occurrence of a service proper, such as printing a given document. For simplicity, we will let these meanings be inferred from context.
supplier	An application or device offering *services* – e.g. a printer.
material	An information asset such as a file or data stream.
capabilities	The set of *services* offered by a *supplier*, or by a whole *environment*.
resources	Are consumed by *suppliers* while providing *services*. Examples are: CPU cycles, memory, battery, bandwidth, etc.
context	Set of human-perceived attributes such as physical location, physical activity (sitting, walking…), or social activity (alone, giving a talk…).
user-perceived state of a task	User-observable set of properties in the *environment* that characterize the support for the task. Specifically, the set of *services* supporting the task, the user-level settings (preferences, options) associated with each of those services, the *materials* being worked on, user-interaction parameters (window size, cursors…), and the user's preferences with respect to quality of service tradeoffs.

Table 2 Summary of the software layers in Aura

layer	*mission*	*roles*
Task Management	***what*** does the user need	– monitor the user's task, context and preferences – map the user's task to needs for services in the environment – complex tasks: decomposition, plans, context dependencies
Environment Management	***how*** to best configure the environment	– monitor environment capabilities and resources – map service needs, and user-level state of tasks to available suppliers – ongoing optimization of the utility of the environment relative to the user's task
Env.	support the user's task	– monitor relevant resources – fine grain management of QoS/resource tradeoffs

computing strategies based on user requirements and on the availability of resources. Moreover, in heavily networked environments, remote servers may constantly change their response times and even availability. Ideally, users should be shielded as much as possible from dealing with such dynamic variations.

Before describing an architecture for supporting user mobility as outlined above, Table 1 clarifies the terminology used throughout this chapter, since although the terms are in common use, their interpretation is far from universal.

Our starting point for supporting user mobility was to design an infrastructure, the **Aura** infrastructure, that exploits knowledge about a user's tasks to automatically configure and reconfigure the environment on behalf of the user. Aura is best explained by a layered view of its infrastructure together with an explanation of the roles of each layer with respect to task suspend-resume and dynamic **adaptation**.

First, the infrastructure needs to know *what* to configure for; that is, what a user needs from the environment in order to carry out his or her tasks. Second, the infrastructure needs to know *how* to best configure the environment: it needs mechanisms to optimally match the user's needs to the capabilities and resources in the environment.

In our architecture, each of these two sub-problems is addressed by a distinct software layer: (1) the *Task Management* layer determines *what* the user needs from the environment at a specific time and location; and (2) the *Environment Management* layer determines *how* to best configure the environment to support the user's needs.

Table 2 summarizes the roles of the software layers in the infrastructure. The top layer, *Task Management* (TM), captures knowledge about user needs and preferences for each activity. Such knowledge is used to coordinate the configuration of the environment upon changes in the user's task or context. For instance, when the user attempts to carry out a task in a new environment, TM coordinates access to all the information related to the user's task, and negotiates task support with Environment Management (EM). Task Management also monitors explicit indications from the user and events in the physical context surrounding the user. Upon getting indication that the user intends to suspend the current task or resume some other task, TM coordinates saving the user-perceived state of the suspended task and recovers the state of the resumed task, as appropriate.

The *EM layer* maintains abstract models of the environment. These models provide a level of indirection between the user's needs, expressed in environment-independent terms, and the concrete capabilities of each environment.

This indirection is used to address both heterogeneity and dynamic change in the environments. With respect to heterogeneity, when the user needs a service, such as speech recognition, EM will find and configure a supplier for that service among those available in the environment. With respect to dynamic change, the existence of explicit models of the capabilities in the environment enables automatic reasoning when those capabilities change dynamically. The Environment Management adjusts such a mapping automatically in response to changes in the user's needs (adaptation initiated by TM), and changes in the environment's capabilities and resources (adaptation initiated by EM). In both cases adaptation is guided by the maximization of a ***utility** function* representing the user's preferences (more in the section on Robustness, below).

The *Environment layer* comprises the applications and devices that can be configured to support a user's task. Configuration issues aside, these suppliers interact with the user exactly as they would without the presence of the infrastructure. The infrastructure steps in only to automatically configure those suppliers on behalf of the user. The specific capabilities of each supplier are manipulated by EM, which acts as a translator for the environment-independent descriptions of user needs issued by TM. Typically, suppliers are developed by wrapping existing applications. Our experience in wrapping over a dozen applications native to Windows and Linux has shown that it is relatively easy to support setting and retrieving the user-perceived state (Balan, Sousa, & Satyanarayanan, 2003; J.P. Sousa, 2005).

This layering offers a clean separation of concerns between what pertains user activities and what pertains the environment. The knowledge about user activities is held by the TM and travels with the user to each environment he or she wishes to carry out activities. The knowledge about the environment stays with the EM and can be used to address the needs of many users.

A significant distinction of this approach to user mobility is that it does not require code or devices to travel with the user. A generic piece of code, Prism, in the TM layer *becomes* an **Aura** for a user by loading models of user activities. Those models are encoded in XML for convenience of mobility across heterogeneous devices (more in the section on Activity Models, below).

Extending to the Home Environment

Although the layered perspective played an important role in clarifying the separation of concerns and protocols of interaction, it captures only the case where users consume services, and software components provide them.

In the smart home domain, software could take responsibility for activities, and users might be asked to contribute services for those activities. For example, a smart home might take charge of the home's security and ask a human to lock the windows when night falls. Other examples of activities include: a user watching a TV show, a user checking on a remote family member, the main hallway facilitating answering the door, and the home keeping a comfortable temperature. These examples prompted us to realize

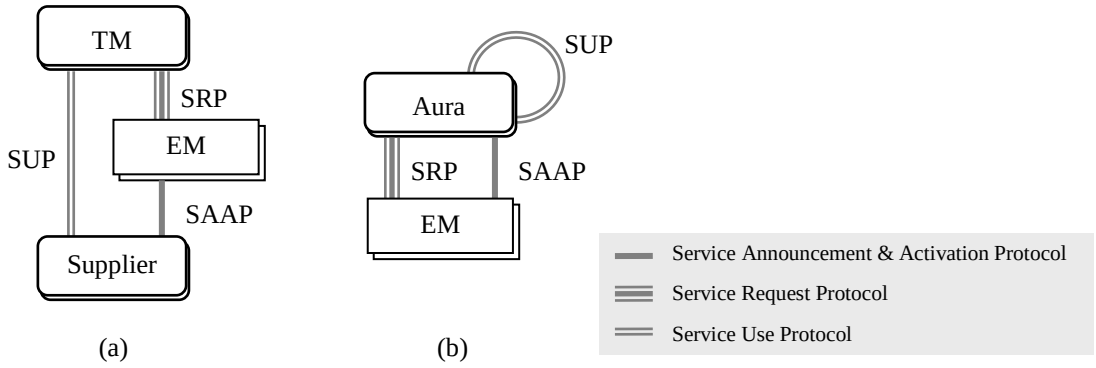

Figure 1 Architectural framework

that any domain entity could have an Aura, and that an Aura might find itself on either the supplying or the consuming side, or both. Specifically, **Auras** can be associated with:

- People: individual residents, or groups, such as a resident's parents, or the entire family.
- Spaces, such as the main hallway, living room or the entire home. Spaces of interest are not necessarily disjoint.
- Appliances, such as a TV, phone, table or couch. Appliances have a well-defined purpose and may have a range of automation levels, from fairly sophisticated (a smart refrigerator), to not automated at all (an old couch).
- Software applications, such as a media player, a video conferencing app, or a people locator. Applications run on general purpose devices, and which applications are available on one such device define the purpose that the device may serve.

Figure 1 shows the run-time architectural framework for an activity-oriented system. The boxes correspond to types of components, and the lines to possible connectors between instances of those types.[1] Part (a) shows our initial understanding, based on the smart office domain, and part (b) the more general framework. Contrasting the two, it is now clear that the TM corresponded to an Aura (of users) that consumed services but supplied none; and suppliers corresponded to Auras of software that supplied services, but consumed none.

In the new architectural framework, when an Aura boots up, it first locates the EM (see the section on Service Discovery, below) and then may engage on the Service Announcement & Activation Protocol (SAAP) to announce any services that its entity provides, as well as on the Service Request Protocol (SRP) to discover services that are relevant to support the entity's activities. Once the services in other Auras are actually recruited by the EM, using the SAAP, the consumer Aura and the supplier Auras interact via the Service Use Protocol (SUP) to reconstruct the user-perceived state of the activity.

Figure 2 shows an example of an architecture that was dynamically created to respond to the needs of a user, Fred, at a particular place, Fred's home. The boxes correspond to run time components (autonomous processes that may be deployed in different boxes) rather than denoting code packaging, and the lines correspond to connectors, that is, actual communication channels that are established dynamically as the components are created. The diagram represents two kinds of components: Auras, with rounded corners,

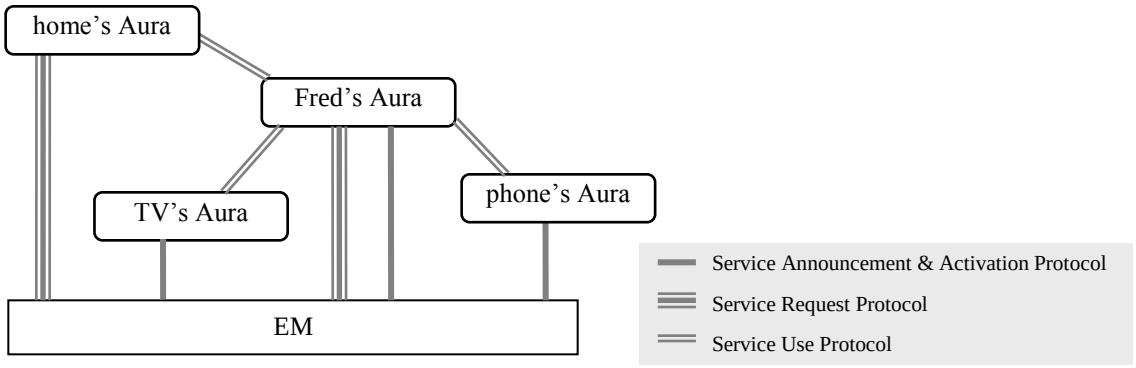

Figure 2 Snapshot of the architecture of one system

and the EM, and it visually identifies the different kinds of interaction protocols between the components (see Figure 1).

The instance of architecture in the example is the result of Fred's Aura interacting with the EM to recruit two suppliers: the TV and the phone's Auras, after interpreting Fred's needs for the desired activity. This architecture may evolve to adapt to changes in Fred's needs, and in the figure, Fred's Aura is also shown as being recruited by the home's Aura to get Fred to open the front door.

Context Awareness

An important decision is how to enable context awareness in activity-oriented computing. Addressing context awareness can be decomposed into three parts: sensing context, distributing contextual information, and reacting to context. We start by discussing the latter.

Potentially, all domain entities, and therefore their Auras, might want to react to context. A user's Aura may change the requirements on an activity, or change which activities are being carried out depending on context such as user location, or whether the user is sitting alone at the office, driving a car, or having an important conversation with his or her boss. Suppliers contributing services to an activity may want to change modalities of those services based on context. For example, an application that shows confidential information may hide that information automatically if someone else is detected entering the room; or an application that requires text or command input may switch to speech recognition if the user needs to use his or her hands, say, for driving a vehicle. The EM may change the allocation of suppliers to for a given activity based on user location. For example, if the user is watching a TV show while moving around the house, different devices may be allocated: the TV in the living room, the computer monitor at the home office, etc. The upshot of this is that contextual information should be accessible to all the boxes in the architecture.

Initially, we thought that a dedicated part in the infrastructure would be in charge of gathering and distributing contextual information: there would be a Context Management component/layer in each environment, just like there is an Environment Management. However, both the contextual information and the policies for distributing such information are associated with each user and not really with the environment where that user happens to be. Therefore, Auras are the hub of knowledge about the entities they represent.

Whenever a component wishes to obtain contextual information about entity X, it will ask X's Aura. X's Aura itself may use a variety of mechanisms to gather information about X. For example, since physical location of a space is normally a non-varying property, the Aura for a home can read the home's location from a configuration file. In contrast, the Aura for an application running on a cell phone equipped with GPS might obtain the application's location from the device's GPS. The Aura for a person P typically obtains P's contextual information via Contextual Information Services (CIS).

Unlike sensors specific to devices or spaces, CIS's are fairly generic. Specifically, devices such as a thermometer attached to a wall, or a window sensor for detecting whether *that* window is open or closed, are accessed only by the Auras of the corresponding physical spaces. In contrast, given a training set with a person's face, a generic face recognizer may be able to track that person's location inside an office building by using cameras spread over rooms and halls.

CISs are integrated into the architecture using the same protocols for discovery and activation as other services; which allows for gracefully handling of activity instantiation in both sensor-rich environments, such as a smart building, and in depleted environments, such as a park.

While CIS components release information based on generic premises such as the rule of law (e.g. only an authenticated Aura for X or a law enforcement agent with a warrant can obtain information about X), the Auras themselves are responsible for knowing and enforcing their entity's privacy policies regulating to whom release which information. As an example of distribution policy, a user may authorize only a restricted group of friends to obtain his or her current location.

Service Discovery

In the initial architectural framework (Figure 1.a,) **service** discovery is coordinated by the Environment Management layer. When we expanded the focus of our research to the smart home domain, around the year 2004, we revisited the problem of service discovery. Among the questions that prompted this revisiting are: are there real advantages in brokering discovery? What are the relevant parameters to guide service discovery (location, quality of service, etc.)? How can discovery be geographically scoped? Would some measure of physical distance be enough for such scoping? Can discovery be scoped by geographical areas that are meaningful to the user? We elaborate on these below.

This first question is what should be the strategy for discovery. Many activities in the smart home domain involve entities performing services for other entities, and it is up to Auras to find and configure the services required by their entities. For example, for watching a TV show, Fred will need some entity to play the video stream for him. Fred's Aura takes care of finding and configuring such an entity in Fred's vicinity (for instance the TV in the living room,) and to change the video stream to other convenient entities whenever Fred moves around the home (a TV in the kitchen, a computer in the office, etc.)

One candidate solution would be to have Auras broadcast service availability and/or service requests. However, service discovery in ubiquitous computing involves not just matching service names or capabilities but, ideally, it would find *optimal* services in

terms of attributes such as desired levels of quality of service, user preferences, proximity, etc. Furthermore, scoping the search would be constrained by network administration policies regarding broadcast (see below). Also, it is hard to establish trust based on broadcast mechanisms.

Because finding the optimal entities to perform services is both a non-trivial problem and common across Auras, there are clear engineering advantages in factoring the solution to this problem out of individual Auras and into a dedicated piece of the infrastructure. Specifically, the benefits of introducing Environment Managers (EMs) as discovery brokers include:

- *Separation of concerns*. It is up to specialized service brokers to know *how* to find the optimal entities to provide services, while each Aura retains the responsibility of knowing *what* services are required by their entity's activities at each moment. By providing a separate locus for optimal discovery in EMs, Auras can focus on the task-specific knowledge required to interact with other Auras, once they are identified.
- *Efficiency*. EMs can cash service announcements, thereby improving the latency of processing a service request, and reducing the network traffic required to locate a service, whenever one is requested.

Auras register the services offered by the entity they represent with an EM. Each service posting includes the type of service and all attributes of the offering entity that characterize the service. For example, Fred's Aura announces that Fred is capable of *answering the door* (service type,) along with Fred's contextual attributes pertaining to his current *location* and whether he is *busy*. Although Fred's Aura might know about Fred's blood pressure, that wouldn't be directly relevant for his ability to answer the door. As another example, the Aura for a printer announces its ability to *print documents*, along with the printer's *location*, *pages per minute* and *queue length* (contextual and quality of service attributes).

Auras may request an EM to find services, as needed by the activities of the entities they represent. Service discovery is guided by optimality criteria in the form of **utility** functions over the attributes of the service suppliers and of the requesting entity. Specifically a service request is of the form:

`find` $\underline{x}$ `: service | y •` $\max u(\underline{p}_x, \underline{p}_y)$

This means: find a set of entities $\underline{x}$, each capable of supplying a `service`, given the requestor entity `y`, such that a utility function `u` over the properties of `y` and of the elements of $\underline{x}$ is maximized. The following are examples with simple utility functions:

Track Fred's location. Upon startup, Fred's Aura issues:

`find` x_1`:people-locating | Fred`

That is, find x_1 capable of providing a people locating service for Fred.

Follow-me video. When Fred wishes to watch a soccer game while moving around the house, his Aura issues:

`find` x_1`:video-playing | Fred • min ®` x_1`.location – Fred.location ™`

That is, find a video player closest to Fred. In this case, maximizing the utility corresponds to minimizing the physical distance between Fred and the video player.

Doorbell. When the doorbell is pressed by someone, the Aura for the main hallway issues:

```
find x1:door-answering, x2:notifying | hallway
     • x1.busy = no & min (® x1.location – hallway.location ™
                           + ® x1.location – x2.location ™)
```

That is, find a notifying mechanism and a door answerer that is not busy and both closest to the hallway and to the notifying mechanism.

Utility functions are quantitative representations of usefulness with respect to each property. Formally, selecting a specific value of a property, such as x_1.busy = no, is encoded as a discrete mapping, specifically:

$$u_{x_1.busy}(yes) \mapsto 0, \quad u_{x_1.busy}(no) \mapsto 1$$

For properties characterized by numeric values, such as the distance to the hallway, we use utility functions that distinguish two intervals: one where the user considers the quantity to be good enough for his activity, the other where the user considers the quantity to be insufficient. *Sigmoid* functions, which look like smooth step functions, characterize such intervals and provide a smooth interpolation between the limits of those intervals. Sigmoids are easily encoded by just two points: the values corresponding to the knees of the curve that define the limits *good* of the good-enough interval, and *bad* of the inadequate interval. The case of "more-is-better" qualities (e.g., *accuracy*) are as easily captured as "less-is-better" qualities (e.g., *latency*) by flipping the order of the *good* and *bad* values (see (João P. Sousa, Poladian, Garlan, Schmerl, & Shaw, 2006) for the formal underpinnings).

In the case studies evaluated so far, we have found this level of expressiveness for utility functions to be sufficient.

Scoping Service Discovery

The second question is how can service discovery be scoped in a way that is meaningful to the user. Specifically, many searches take place in the user's immediate vicinity, such as the user's home.

However, neither physical distance nor network range are good parameters to scope discovery. For example, if the user's activity asks for a device to display a video, the TV set in the apartment next door should probably not be considered, even though it might be just as close as other candidates within the user's apartment, and be within range of the user's wireless network as well. To be clear, once a set of devices is scoped for discovery, then physical distance may be factored in as a parameter for *optimality* (see above).

Furthermore, sometimes users may want to scope discovery across areas that are not contiguous. For example, suppose that Fred is at a coffee shop and wants to print an interesting document he just found while browsing the internet. Fred may be willing to have the document printed either at the coffee shop, or at Fred's office, since Fred is heading there shortly. A printer at a store down the street may not be something that Fred would consider, even though it is physically closer than Fred's office.

The question about scoping service discovery can then be refined into (a) if not by distance or network boundaries, how can the range of one environment be defined? and (b) how to coordinate discovery across non-contiguous environments?

When an Aura directs a discovery request to an EM, by default, discovery runs across all services registered with that EM. That is, the range of an environment is defined by the services that registered with its EM. The question then becomes, how does an Aura know with which EM it should register its services with? For example, how would the Aura for the TV set in Fred's living room know to register its services with the EM in Fred's apartment, and not with the neighbor's?

Auras resolve their physical location into the network address of the appropriate EM by using the Environment Manager Binding Protocol (EMBP).[2] This service plays a similar role to the Domain Naming Service in the internet, which resolves URI/URLs into the network address of the corresponding internet server. Physical locations are encoded as Aura Location Identifiers (ALIs), which structure and intent is similar to Universal Resource Identifiers (URIs) in the internet. Like URIs, ALIs are a hierarchical representation mean to be interpreted by humans and resolved automatically. For example, *ali://pittsburgh.pa.us/zip-15000/main-street/1234/apt-6* might correspond to Fred's apartment; and *ali://aura.cmu.edu/wean-hall/floor-8/8100-corridor* to a particular corridor on the 8th floor of Wean Hall at Carnegie Mellon University.

Requests for discovery across remote and/or multiple environments can be directed to the local EM, which then coordinates discovery with other relevant EMs (more below). The following are examples of such requests. When Fred is at home and wishes to print a document at the office, his Aura would issue a request like

```
find x:printing | Fred • u(…)
      @ ali://aura.cmu.edu/wean-hall/floor-8/8100-corridor
```

Or, if Fred wanted to consider alternatives either at home or at his office:

```
find x:printing | Fred • u(…)
      @ ali://aura.cmu.edu/wean-hall/floor-8/8100-corridor,
        ali://pittsburgh.pa.us/zip-15000/main-street/1234/apt-6
```

Or, if Fred wanted to search a number of adjacent environments, such as all the environments in his office building:

```
find x:printing | Fred • u(…)
      @ ali://aura.cmu.edu/wean-hall
```

Any such requests are directed by the requestor Aura to the local EM, which then resolves such requests in three steps:

1. Use the EMBP to identify the EMs that cover the desired region.
2. Obtain from such EMs all the service descriptions that match the requested service types.
3. Run the service selection algorithms over the candidate set of services.

Activity Models

What to include in activity models is ultimately determined by the purpose that those models are meant to serve. In some applications of activity models the goal is to assist users with learning or with performing complex tasks. Examples of these are applications

```
Task  ::= auraTask: id;
      Prefs {ServiceSnapshot | MaterialSnapshot | Config}

ServiceSnapshot ::= service: id type;
      Settings
MaterialSnapshot ::= material: id;
      State

Config ::= configuration: name weight;
      { Service | Connection }

Service ::= service: id;
      {Uses}
Uses  ::= uses: materialId;

Connection ::= connection; id type;
      Attach QoSPrefs
Attach ::= attach: ;
      From To
From  ::= from: serviceId port;
To    ::= to: serviceId port;
```

Figure 3 Grammar for specifying task models.

to automated tutoring, expert systems to help engineers repair complex mechanisms, such as trains and airplanes, and automated assistants to help manage complex activities such as organizing a conference (Garlan & Schmerl, 2007; Siewiorek, 1998). For these kinds of applications, models of activities may include a specification of workflow, as a sequence of steps to be performed, and cognitive models of the user.

In the smart office domain, we experimented with enabling users to suspend their ongoing activities and resume them at a later time and/or at another location, possibly using a disjoint set of devices. For that purpose, the models capture user needs and preferences to carry out each activity. Specifically, such models include of a snapshot of the services and materials being used during the activity, as well as utility theory-based models of user preferences (for details on the latter, see (João P. Sousa et al., 2006)).

Figure 3 shows a grammar for modeling **activities**, or tasks, as a set of possibly interconnected services. This grammar follows a variant of the Backus-Naur Form (BNF, see for instance (ISO, 1996)). To simplify reading the specification, we drop the convention of surrounding non-terminal symbols with angle brackets, and since the task models are built on top of XML syntax, we augment the operators of BNF with the following:

```
E ::= t: A; C
```

defines a type E of XML elements with tag t, attributes A, and children C, where t is a terminal symbol, A is an expression containing only terminals (the attribute names), and C is an expression containing only non-terminals (the child XML elements). In this restricted use of BNF, whether a symbol is a terminal or non-terminal is entirely established by context. So, for instance the rule

```
Book = book: year ISBN; Title {Author}
```

allows the following as a valid element:

```
<book year=”2004” ISBN=”123”>
  <title>...</title>
  <author>...</author>
  <author>...</author>
</book>
```

Specifically, in Figure 3, a task (model) is an XML element with tag `auraTask`, with one `id` attribute, and with one `Prefs` child, followed by an arbitrary number of `ServiceSnapshot`, `MaterialSnapshot`, and `Config` children. A task may be carried out using one of several alternative service configurations of services.

```
<auraTask id="34">
  <preferences>
    <service template="default" id="1"/>
    <service template="default" id="2"/>
  </preferences>
  <service type="play Video" id="1">
    <settings mute="true"/>
  </service>
  <material id="11">
    <state>
      <video state="stopped" cursor="0"/>
      <position xpos="645" ypos="441"/>
      <dimension height="684" width="838"/>
    </state>
  </material>
  <service type="edit Text" id="2">
    <settings>
      <format overtype="0"/>
      <language checkLanguage="1"/>
    </settings>
  </service>
  <material id="21">
    <state>
      <cursor position="31510"/>
      <scroll horizontal="0" vertical="7"/>
      <zoom value="140"/>
      <spellchecking enabled="1" language="1033"/>
      <window height="500" xpos="20" width="600" mode="min" ypos="100"/>
    </state>
  </material>
  <configuration name="all" weight="1.0">
    <service id="2">
      <uses materialId="21"/>
    </service>
    <service id="1">
      <uses materialId="11"/>
    </service>
  </configuration>
  <configuration name="only video" weight="0.7">
    <service id="1">
      <uses materialId="11"/>
    </service>
  </configuration>
</auraTask>
```

Figure 4 Example task model for reviewing a video clip.

Services stand for concepts such as *edit text*, or *browse the web*, and materials are files and data streams manipulated by the services. A service may manipulate zero or many materials; for instance, text editing can be carried out on an arbitrary number of files simultaneously. That relationship is captured by the `Uses` clauses within the `Service` element.

The snapshot of the user-perceived state of the task is captured in the *`Settings`* and *`State`* elements. The *`Settings`* element captures the state that is specific to a service, and shared by all materials manipulated by that service, while the *`State`* element captures the state that is specific to each material. A detailed discussion of this grammar can be found in (J.P. Sousa, 2005).

Figure 4 shows one example of a task model for reviewing a video clip, which formally is a sentence allowed, or generated, by the grammar in Figure 3. This example was captured while running the infrastructure described in the section on System Design. The user defined two alternative configurations for this task: one including both playing the video and taking notes, the other, playing the video alone. Both services use a single material: *play video* uses a video file, with material id 11, and *edit text* uses a text file, with material id 21. The user-perceived state of the task is represented as the current service settings, under each service, and the current state of each material. For instance, the state of the video includes the fact that the video is stopped at the beginning (the cursor is set to 0 time elapsed), and it indicates the position and dimensions of the window showing the video.

Extending to the Home Environment

In the smart home domain, in addition to supporting suspend/resume of activities, we wanted to enable users to delegate responsibility for some activities to **Auras**. Examples of the latter activities include managing intrusion detection for the home, finding a person to answer the door for a visitor, or assisting with monitoring elder family members.

The research questions then become: is the services and materials view of activities adequate in the smart home domain? For enabling Auras to take responsibility for activities, which concepts should activity models capture?

The usefulness of capturing the services needed for an activity seems to carry well into the smart home domain. For example, in the case of the doorbell scenario, the activity of answering the door requires finding services such as notification, can be supplied by devices such as a telephone, a TV, a buzzer, etc., and *door answerer*, which can be supplied by a qualified person (e.g., not a toddler). Selecting the suppliers for such services is guided by the home owner's preferences encoded in the activity model; which may include things such as: the door needs to be answered within a certain time, and that the notification service should be in close proximity to the candidate door answerer.

A prototype of this case study has shown that these models can handle sophisticated policies of configuration (e.g., excluding children from answering the door, or specifying criteria for proximity) and that they trivially accommodate the dynamic addition of new notification devices.

This prototype also highlighted two fundamental differences between the kinds of activities supported in the smart-office domain and the ones we target in the smart-home domain. The first difference is that, while in the office domain services were only

provided by automated agents (software), now people may also be asked to provide services. This has implications on how Auras control service supply, since people are much more likely than software to do something totally different than what they are being asked. In the example, after being notified to answer the door, a person may get sidetracked and forget about it. It is up to the responsible Aura to monitor whether or not the service is being delivered, and react to a "fault" in a similar way as it would in the case of faulty software: by seeking a replacement (more in the section on Robustness).

The second difference is that, in the smart home domain, Auras may take the responsibility for activities: and this is related to the question above of which concepts to capture in activity models to enable that to happen. In the smart office domain, when a fault cannot be handled, for example, if a suitable replacement cannot be found for a faulty supplier, the problem is passed up to the entity responsible for the activity, i.e. the user. If an Aura is to be truly responsible for an activity, it must be take charge of such situations as well.

One way of addressing a hurdle in one activity, is to carry out another activity that circumvents the hurdle. In the example, if the hallway Aura cannot find a person to answer the door, it may take a message from the visitor, or initiate a phone call to the person being visited.

A simple enhancement of activity models to allow this is to support the specification of conditions to automatically resume or suspend activities. Such conditions are expressed as Boolean formulas over observation of contextual information. For example, if everyone left the house, resume the intrusion detection activity.

For these models to cover situations as the one where a person could not be found to answer the door, contextual information needs to be rich enough to include semantic observations, such as "the door could not be open for a visitor."

Another scenario where we tested this approach is the elder care scenario. The Aura for Susan, Fred's grandmother, runs a perpetual task that recruits a heart monitor service for her. Susan defined under which conditions her Aura should trigger the task of alerting the family. When defining such conditions, Susan takes into consideration her physician's recommendations, but also conditions under which she may desire privacy. Fred's Aura runs a perpetual task of monitoring contextual postings by Susan's Aura. It is up to John to (a) define that posting such a notification should trigger the task of alerting him, and (b) define the means employed by his Aura to carry out such a task. For example, if Fred is at the office, his Aura sends an instant message to Fred's computer screen; otherwise, it sends a text message to Fred's cell phone.

While these are simple scenarios, they illustrate the ability to chain activities, and to direct the exact behavior of activities, by capturing conditions on contextual information in the models of activities. Such conditions are associated to the operations of either resuming or suspending activities, and can be monitored by Auras to automatically initiate the corresponding operation.

Formally, condition-action primitives can be used to express the same space of solutions than other more sophisticated approaches, such as models of activities based on workflow notations, or on hierarchical decomposition of activities. Which approach would be more suitable for end-users to express and understand such models is an open research problem.

Robustness

The term *robustness* in activity-oriented computing is interpreted very broadly: is the system's behavior consistent with the users' expectations, even under unanticipated circumstances. In this section, we first use the examples in the Background section to identify key robustness requirements. We then look at the challenges associated with supporting robust operation, distinguishing between general challenges and challenges that are specific to the home environment. Finally, we summarize some results showing how we support robust tasks in an office environment and discuss how these results can be extended to support activities in the home.

Properties

In daily use, the system should correctly identify the users' intent and should support a wide variety of activities in a way that is consistent with their preferences and policies. If users observe unexpected behavior, the system should be able to explain its behavior. This will increase the users' confidence in the system and will allow the system to improve over time. For example, by adjusting preferences and policies, either manually by the user or automatically by the system (case-base reasoning) the system's future behavior can be made to better match user intent. Similarly, the system should be able to engage users if input is confusing or unexpected. Ideally, the system would be able to recognize undesirable or unsafe actions, e.g. a child opening the door for a stranger.

The above properties must also be maintained as the system evolves and under failure conditions. For example, when new services or devices are added (e.g. camera and face recognition software is added to support the doorbell scenario) or become unavailable (e.g. the license for the face recognition software expired), the system should automatically adapt to the available services.

Challenges

When we looked at how to support user activities and tasks in different environments (e.g., work in an office, daily activities in the home, or guiding visitors in a museum) we found that several key challenges are shared across these environments. These generic challenges include capturing and representing user intent, discovering and managing services and devices (suppliers), and optimizing resources allocation to maximize overall system utility. All these functions should be adaptive, i.e. automatically adapt to changes in the computational and physical environment and to changes in the goals and preferences of users.

Each environment also adds its own challenges. For example, activities in homes are device-centric (e.g., displays, sound) or include physical actions that involve people (e.g., opening doors). Managing and allocating such "resources" is very different from an office environment, where tasks are computer-centric and are supported by executing applications that use a variable amount of resources (network bandwidth, CPU, battery power). Similarly, the interactions with users are very different in the home (discreet interface for non-experts) and the office (keyboard/mouse/display used by computer knowledgeable users).

Robustness in an Office Environment

In order to achieve robustness in a smart-office environment, we have designed, implemented and evaluated an infrastructure that uses utility theory to dynamically select the best achievable configuration of services, even in the face of failures and coming online of better alternatives (João P. Sousa et al., 2006).

Robustness is achieved through self-**adaptation** in response to events ranging from faults, to positive changes in the environment, to changes in the user's task. Self-adaptation is realized through a closed-loop control system design that senses, actuates, and controls the runtime state of the environment based on input from the user. Each layer reacts to changes in user tasks and in the environment at a different granularity and time-scale. Task Management acts at a human perceived time-scale (minutes), evaluating the adequacy of sets of services to support the user's task. Environment Management acts at a time-scale of seconds, evaluating the adequacy of the mapping between the requested services and specific suppliers. Adaptive applications (fidelity-aware and context-aware) choose appropriate computation tactics at a time-scale of milliseconds.

Let us illustrate the behavior of the system using the following scenario. Fred is engaged in a conversation that requires real-time speech-to-speech translation. For that task, assume the Aura infrastructure has assembled three services: speech recognition, language translation, and speech synthesis. Initially both speech recognition and synthesis are running on Fred's handheld. To save resources on Fred's handheld, and since language translation is computationally intensive, but has very low demand on data-flow (the text representation of each utterance), the translation service is configured to run on a remote server. We now discuss how the system adapts in response to faults, variability in resource and service availability, and changes in the user's task requirements.

Fault tolerance. Suppose now that there is loss of connectivity to the remote server, or equivalently, that there is a software crash that renders it unavailable. Live monitoring at the EM level detects that the supplier for language translation is lost. The EM looks for an alternative supplier for that service, e.g., translation software on Fred's handheld, activates it, and automatically reconfigures the service assembly.

Resource and fidelity-awareness. Computational resources in Fred's handheld are allocated by the EM among the services supporting Fred's task. For computing optimal resource allocation, the EM uses each supplier's spec sheet (relating fidelity levels with resource consumption), live monitoring of the available resources, and the user's preferences with respect to fidelity levels. Resource allocation is adjusted over time. For example, suppose that during the social part of the conversation, Fred is fine with a less accurate translation, but response times should be snappy. The speech recognizer, as the main driver of the overall response time, gets proportionally more resources and uses faster, if less accurate, recognition algorithms (Balan et al., 2003).

Adaptation is also needed to deal with changes in resource availability. Each supplier issues periodic reports on the Quality of Service (QoS) actually being provided – in this example, response time and estimated accuracy of recognition/translation. Suppose that at some point during the conversation, Fred brings up his calendar to check his availability for a meeting. The suppliers for the speech-to-speech translation task, already stretched for resources, reduce their QoS below what Fred's preferences state as acceptable. The EM detects this "soft fault", and replaces the speech recognizer by a lightweight

component, that although unable to provide as high a QoS as the full-fledged version, performs better under sub-optimal resource availability. Alternatively, suppose that at some point, the language translation supplier running on the remote server (which failed earlier) becomes available again. The EM detects the availability of a new candidate to supply a service required by Fred's task, and compares the estimated utility of the candidate solution against the current one. If there is a clear benefit, the EM automatically reconfigures the service assembly. In calculating the benefit, the EM factors in a cost of change. This mechanism introduces hysteresis in the reconfiguration behavior, thus avoiding oscillation between closely competing solutions.

Task requirements change. Suppose that at some point Fred's conversation enters a technical core for which translation accuracy becomes more important than fast response times. The TM provides the mechanisms to allow Fred to quickly indicate his new preferences; for instance, by choosing among a set of preference templates. The new preferences are distributed by the TM to the EM and all the suppliers supporting Fred's task. Given a new set of constraints, the EM evaluates the current solution against other candidates, reconfigures, if necessary, and determines the new optimal resource allocation. The suppliers that remain in the configuration, upon receiving the new preferences, change their computation strategies dynamically; e.g., by changing to algorithms that offer better accuracy at the expense of response time.

Suppose that after the conversation, Fred wants to resume writing one of his research papers. Again, the TM provides the mechanisms to detect, or for Fred to quickly indicate, his change of task. Once the TM is aware that the conversation is over it coordinates with the suppliers for capturing the user-level state of the current task, if any, and with the EM to deactivate (and release the resources for) the current suppliers. The TM then analyses the description it saved the last time Fred worked on writing the paper, recognizes which services Fred was using and requests those from the EM. After the EM identifies the optimal supplier assignment, the TM interacts with those suppliers to automatically recover the user-level state where Fred left off. See (J. P. Sousa & Garlan, 2003) for a formal specification of such interactions.

Extending to the Home Environment

We are currently enhancing this solution to provide robust support for activities in the home. While the key challenges are the same (e.g. optimizing utility, adapting to changes...) extensions are needed in a number of areas.

First, activities in the home are very different from tasks in the office. For example, since some activities in the home involve physical actions, people must be involved (e.g., open a door), i.e. people become suppliers of services. Moreover, some tasks are not associated with individuals, but with the home itself (e.g., responding to the doorbell or a phone call). This change in roles means that it is even more critical to make appropriate allocations since the cost of mistakes is much higher, e.g., people will be much less willing to overlook being personally inconvenienced by a wrong decision, than when a suboptimal application is invoked on their computer.

Second, many activities in the home involve the use of devices that are shared by many people, or involve deciding who should perform a certain action. This means that the Task Manager will typically need to balance the preferences and goals of multiple users. An extreme example is conflicts, e.g., when multiple users would like to use the

same device. In contrast, tasks in the office typically involve only personal resources (e.g., a handheld) or resources with simple sharing rules (e.g., a server).

Third, the methods for interaction with the system will be much different in the home. Even on a handheld, Fred had access to pull down menus a keyboard to reliably communicate with the system. For the home environment, we are exploring natural modalities of interaction, which are less intrusive, but more ambiguous (more in the section on Future Research).

Finally, uncertainty will play a more significant role in the home, e.g. because of unpredictably behavior when people are asked to perform services, or due to ambiguity caused by primitive I/O devices. Work in progress is extending the utility optimization components to explicitly consider uncertainty.

FUTURE RESEARCH

Some of the challenges identified in this chapter are the topic of undergoing and future work, such as research on the kinds of knowledge to capture in activity models so to support mixed-initiative control, including delegation and collaboration among human and automated agents. Below we summarize our current work on human-computer interaction and on security and privacy for AoC systems.

User Interfaces for Managing Activities

Human-computer interaction in the office domain currently uses one de-facto standard modality, based on keyboards, pointing devices, and windows-based displays. In a more general ubiquitous computing setting, natural modalities such as speech and gesture may be highly desirable, but they also may lead to ambiguity and misunderstanding. For example, if Fred points at a TV where a soccer game is playing and leaves the room, does that mean that Fred wants to keep watching the game while moving around the house, that the TV should pause the game until Fred returns, or that the TV should be turned off?

Rather than trying to pick a privileged modality of interaction, we take the approach that interactions between humans and Auras may have many channels that complement and serve as alternatives to each other. For example, users might indicate their intention to suspend an activity verbally, but might sometimes prefer a graphical interface to express a sophisticated set of contextual conditions for when an activity should be automatically resumed. The research questions then become: what are appropriate modalities for each kind of interaction? Is there a role for explicit interactions, as well as for implicit interactions based on sensing and inference? Can different modalities be coordinated, contributing to disambiguate user intentions? What mechanisms can be used to detect and recover from misunderstandings? What are specific technologies that can be harnessed in the home?

To support explicit interactions, we started exploring technologies such as Everywhere Displays and RFID. The Everywhere Displays technology uses a digital camera to track down the location of a user, and then uses a projector to project an image of the interface onto a surface near the user (Kjeldsen, Levas, & Pinhanez, 2004). The feedback loop through the camera allows the image to be adjusted for certain characteristics of the surface, such as color and tilt. The user interacts with this image by performing hand motions over the image, which are then recognized via the camera. This technology

supports a metaphor similar to the point-and-click metaphor, although fewer icons seem to be feasible relative to a computer screen, and a rich set of command primitives, such as double clicking or selecting a group of objects, seems harder to achieve.

RFID technology supports a simple form of tangible interfaces (Greenberg & Boyle, 2002). For example, RFID tags can be used to create tangible widgets for activities. In the example where Fred is watching the game on TV, Fred may bind an activity widget with the show playing on the TV by swiping the widget near the TV. That activity may be activated in other rooms by swiping the activity widget by a reader in the room, or deactivated it by swapping the widget again, once activated (see demo video at (J.P. Sousa, Poladian, & Schmerl, 2005)).

Tradeoff between Ubiquity and Security

The big question to be answered is: can ubiquity be reconciled with goals of security and privacy? There seems to be tradeoff between the openness of ubiquitous computing and security assurances. The very meaning of *ubiquity* implies that users should be enabled to use the services offered by devices embedded in many different places. But how confident can users be that those devices, or the environment where they run, will not take advantage of the access to the user's information to initiate malicious actions?

Rather that taking an absolute view of security and privacy, we argue that there are different requirements for different activities. For example, the computing environment at a coffee shop could be deemed unsafe to carry out online financial transactions, but acceptable for sharing online vacation photos with a friend.

In essence, this is a problem of controlling access: ideally, a ubiquitous computing environment would gain access only to the information pertaining to the activities that a user is willing to carry out in that environment, and none other.

Unfortunately, existing solutions for controlling access are not a good fit to this problem because they make a direct association between identity and access. Specifically, once a user authenticates, he gains access to *all* the information and resources he is entitled to, and so does the computing environment where the user authenticated.

A candidate solution would be to associate access control to the cross-product of users and environments: in the example, user Fred at the coffee shop would get access to a limited set of activities, but user Fred at his office would get access to a wide range (possibly all) of Fred's activities. A serious problem with this solution is that it would require the pre-encoding of all the types of environments where the user might want to access his or her activities.

Another candidate solution would be for users to have multiple identities: Fred at the coffee shop would use an identity that has access to the vacation photos, but not to online banking. This solution has two obvious problems: first, separating the activities and associated information for the different identities may not be clear cut, and may quickly become cumbersome for moderately high numbers of activities. Second, if users are to be given the freedom to define new identities and the corresponding access control, does that mean that every user should be given security administration privileges?

We are currently investigating an access control model centered on the notion of *persona*. A user is given one identity and may define multiple personae associated with

that identity. The user may freely associate activities with personae in a many-to-many fashion, and may also define which credentials are required to activate each persona. This model has a number of benefits, as follows.

First, it allows users to manage which activities are seen by an arbitrary environment (by authenticating specific personae) while drawing a clear boundary on the administrative privileges of each user.

Second, users may draw on rich forms of authentication to make the overhead of authentication proportionate to the security requirements. For example, for activating Fred's financial persona, Fred may require two forms of id to be presented, such as entering a password *and* scanning an id card, while for his social persona, a weak form of authentication, such as face or voice recognition, will suffice.

Third, the model offers users a coherent view of the personal workspace centered on their identity, while enabling users to expand the set of accessible activities at will, by providing the credentials required to activate the desired personae.

CONCLUSION

The key idea of Activity-oriented Computing (AoC) is to capture models of user activities and have systems interpret those models at run time. By becoming aware of what user activities entail, systems can do a better job at supporting those activities, either by facilitating access to the activities while relieving users from overhead such as configuring devices and software, or by taking responsibility for parts or whole activities.

This chapter described the authors' work on building systems to support AoC. It discussed how those systems may address challenges inherent to the problem domain, such as user mobility and conflict resolution, as well as challenges that are entailed by building the systems themselves. Specifically, (a) defining what to capture in activity models (b) designing systems that do a good job at supporting user activities while addressing the challenges in the problem domain, and (c) making those systems robust, self-aware, self-configurable, and self-adaptable. The chapter dissected those challenges, identified specific research questions, and described how the authors answered these questions for the past six years, as their understanding of the issues improved.

The main contributions of this work are as follows:

- Pragmatic models of user activities that enable mobile users to instantiate activities in different environments, taking advantage of diverse local capabilities without requiring the use of mobile devices, and retaining the ability to reconstitute the user-perceived state of those activities.
- Mechanisms that enable scoping service discovery over geographical boundaries that are meaningful to users, and which can be specific to each activity and be freely defined.
- A utility-theoretic framework for service discovery that enables optimization of sophisticated, service-specific models of QoS and context properties.
- A robustness framework, based on the same utility-theoretic framework, that departs from the traditional binary notion of fault and uniformly treats as an optimization problem faults, "soft faults" (unresponsiveness to QoS requirements,) and conflicts in accessing scarce resources.

- Closed-loop control that enables systems to become self-aware and self-configurable, and to adapt to dynamic changes in both user/activity requirements and environment resources.
- A software architecture that harmoniously integrates all the features above, additionally (a) integrating context sensing according to the capabilities of each environment, and (b) coordinating adaptation policies with applications that may contain their own fine-grain mechanisms for adaptation to resource variations.

This chapter also summarized ongoing and future work towards addressing other challenges, namely, supporting activities where human and automated agents collaborate (mixed-initiative activities), exploring human-computer interaction modalities for AoC in ubiquitous computing environments, and investigating models for security and privacy.

REFERENCES

Abowd, G., Bobick, A., Essa, I., Mynatt, E., & Rogers, W. (2002). *The Aware Home: Developing Technologies for Successful Aging.* Paper presented at the AAAI Workshop on Automation as a Care Giver, Alberta, Canada.

Arnstein, L., Sigurdsson, S., & Franza, R. (2001). Ubiquitous Computing in the Biology Laboratory. *Journal of Lab Automation (JALA), 6*(1), 66-70.

Balan, R. K., Sousa, J. P., & Satyanarayanan, M. (2003). *Meeting the Software Engineering Challenges of Adaptive Mobile Applications* (Tech. Report No. CMU-CS-03-111). Pittsburgh, PA: Carnegie Mellon University.

Brumitt, B., Meyers, B., Krumm, J., Kern, A., & Shafer, S. (2000). EasyLiving: Technologies for intelligent environments. In Gellersen, Thomas (Eds) *2nd Int'l Symposium on Handheld and Ubiquitous Computing* (HUC2000), LNCS 1927, pp. 12-29, Bristol, UK: Springer-Verlag.

Christensen, H., & Bardram, J. (2002, September). Supporting Human Activities – Exploring Activity-Centered Computing. In Borriello and Holmquist (Eds.) *4th International Conference on Ubiquitous Computing* (UbiComp 2002), LNCS 2498, pp. 107-116, Göteborg, Sweden: Springer-Verlag.

Garlan, D., & Schmerl, B. (2007). The RADAR Architecture for Personal Cognitive Assistance. *International Journal of Software Engineering and Knowledge Engineering, 17*(2), in press.

Greenberg, S., & Boyle, M. (2002). Customizable physical interfaces for interacting with conventional applications. In *15th Annual ACM Symposium on User Interface Software and Technology* (UIST 2002), pp. 31-40, ACM Press.

Intille, S. (2002). Designing a home of the future. *IEEE Pervasive Computing, 1*(2), 76-82.

ISO. (1996). *Extended Backus-Naur Form* (No. ISO/IEC 14977:1996(E)). www.iso.org: International Standards Organization.

Kjeldsen, R., Levas, A., & Pinhanez, C. (2004). Dynamically Reconfigurable Vision-Based User Interfaces. *Journal of Machine Vision and Applications, 16*(1), 6-12.

Kozuch, M., & Satyanarayanan, M. (2002). *Internet Suspend/Resume*. Paper presented at the 4th IEEE Workshop on Mobile Computing Systems and Applications, available as Intel Research Report IRP-TR-02-01.

Logitech, I. Logitech Harmony Remote Controls. from http://www.logitech.com

MacIntyre, B., Mynatt, E., Voida, S., Hansen, K., Tullio, J., & Corso, G. (2001). Support for Multitasking and Background Awareness Using Interactive Peripheral Displays. In *ACM User Interface Software and Technology* (UIST'01), pp. 41-50, Orlando, FL.

Ponnekanti, S., Lee, B., Fox, A., & Hanrahan, P. (2001). ICrafter: A Service Framework for Ubiquitous Computing Environments. In Abowd, Brumitt, Shafer (Eds) *3rd Int'l Conference on Ubiquitous Computing* (UbiComp 2001), LNCS 2201, pp. 56-75. Atlanta, GA: Springer-Verlag.

Richardson, T., Bennet, F., Mapp, G., & Hopper, A. (1994). A ubiquitous, personalized computing environment for all: Teleporting in an X Windows System Environment. *IEEE Personal Communications Magazine, 1*(3), 6-12.

Rochester, U. The Smart Medical Home at the University of Rochester. from http://www.futurehealth.rochester.edu/smart_home

Román, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R., & Narhstedt, K. (2002). Gaia: A Middleware Infrastructure for Active Spaces. *IEEE Pervasive Computing, 1*(4), 74-83.

Siewiorek, D. (1998). Adtranz: A Mobile Computing System for Maintenance and Collaboration. In *2nd IEEE Int'l Symposium on Wearable Computers*, pp. 25-32: IEEE Computer Society.

Sousa, J. P. (2005). *Scaling Task Management in Space and Time: Reducing User Overhead in Ubiquitous-Computing Environments* (Tech. Report No. CMU-CS-05-123). Pittsburgh, PA: Carnegie Mellon University.

Sousa, J. P., & Garlan, D. (2003). *The Aura Software Architecture: an Infrastructure for Ubiquitous Computing* (Tech. Report No. CMU-CS-03-183). Pittsburgh, PA: Carnegie Mellon University.

Sousa, J. P., Poladian, V., Garlan, D., Schmerl, B., & Shaw, M. (2006). Task-based Adaptation for Ubiquitous Computing. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, Special Issue on Engineering Autonomic Systems, 36*(3), 328-340.

Sousa, J. P., Poladian, V., & Schmerl, B. (2005). Project Aura demo video of the follow me scenario. from http://www.cs.cmu.edu/~jpsousa/research/aura/followme.wmv

Sycara, K., Paolucci, M., Velsen, M. v., & Giampapa, J. (2003). The RETSINA MAS Infrastructure. *Joint issue of Autonomous Agents and MAS, Springer Netherlands, 7*(1-2), 29-48.

Wang, Z., & Garlan, D. (2000). *Task Driven Computing* (Tech. Report No. CMU-CS-00-154). Pittsburgh, PA: Carnegie Mellon University.

[1] For generality, the protocols of interaction were renamed from previous architecture documentation (e.g. [**Error! Reference source not found.**,**Error! Reference source not found.**]). For instance, the EM – Supplier protocol is now the Service Announcement and Activation Protocol (SAAP). Since these

protocols were already based on peer-to-peer asynchronous communication, no changes were implied by the transition to the new perspective of the architectural framework.

[2] As discussed in the subsection on context awareness, there is a variety of mechanisms for Auras to obtain their physical location, or more precisely, the location of their corresponding entities.