Towards Ubiquitous Computing Applications Composed from Functionally Autonomous Hybrid Artifacts

Nicolas Drossos[1], Irene Mavrommati[1], and Achilles Kameas[1,2]

[1] Computer Technology Institute, Patras, Greece
[2] Hellenic Open University, Greece

1 Introduction

People are an intrinsic part of a Disappearing Computer environment; it is their actions and behavior, their wishes and needs that shape the environment. People have always been building "ecologies" in their living spaces, by selecting objects and then arranging them in ways that best serve their activities and their self-expression. According to the Ambient Intelligence (AmI) vision (ISTAG 2006) people will be able to build more advanced "ecologies", also known as *UbiComp applications*, by configuring and using "augmented" objects; these objects may be totally new ones or updated versions of existing ones. An important new aspect of *AmI environments* is the merging of physical and digital spaces, i.e. tangible objects and physical environments are acquiring digital representations. The traditional computer disappears in the environment, as the everyday objects in it become augmented with Information and Communication Technology (ICT) components (i.e. sensors, actuators, processor, memory, communication modules, etc.) and can receive, store, process and transmit information, thus becoming *AmI objects*.

AmI objects are considered to extend the functionality of physical objects, which nevertheless still maintain their physical properties and natural uses i.e. an "augmented desk" is still a desk that can be used as such without any dependencies from other objects or environmental factors. The artifacts' dependence on a source of power is another factor that determines their autonomy, but it will not be dealt with in this chapter.

When thinking of UbiComp applications for everyday life, it comes out that almost every object, be it physical or software, portable or stationary, resource constrained or not, can potentially become an AmI object and participate in such applications. This means that AmI objects may have to exchange data if combined in the same application, although they may be created by various manufacturers, using various, often proprietary software or hardware technologies or protocols. This heterogeneity factor constitutes a key challenge for systems aiming to support the composition of UbiComp applications.

Another important challenge is the dynamic nature of UbiComp applications deriving both from the dynamic and ad-hoc way that people use UbiComp applications and from the fact that AmI objects are susceptible to failures (e.g. hardware, communication etc.). Thus UbiComp applications need to be *easy-to-use* for a wide range of users, *adaptive* to contextual changes, *robust* enough and *fault-tolerant*. *Scalability* is also a tough issue, since UbiComp applications usually involve a large number of participating AmI objects.

N. Streitz, A. Kameas, and I. Mavrommati (Eds.): The Disappearing Computer, LNCS 4500, pp. 161 – 181, 2007.

In the end, the human element can be catalytic for the whole system: one of the factors that can cause "emerging" and previously unforeseen functionality is the inherent human creativity and the human capability for problem solving and expression. Nevertheless, this relies on people's willingness to adopt ubiquitous computing technology and their understanding of it, the technology's learnability, flexibility and openness for adaptation.

Adoption depends on understanding, which is a two-way channel between two seemingly distant terrains: technology providers have to understand people's needs and wants; people have to understand the capabilities, limitations and use of the new technology. The research that is described in this chapter aims to create a conceptual framework that will increase the bandwidth of the "understanding channel" and provide the technological platform and tools that will support the development of Ubiquitous computing applications.

2 An Overview of Existing UbiComp Approaches

Building and supporting the operation of UbiComp applications is a demanding task, as it comes up against a number of challenges inherent in distributed systems; more constraints are imposed, new parameters are introduced, a wider user community is targeted, to mention but some. Some of the major requirements a UbiComp system has to confront are: mask the *heterogeneity* of networks, hardware, operating systems etc.; tackle *mobility* and *unavailability* of nodes; support component *composition* into applications; *context awareness*; preserve object *autonomy* even for *resource constraint* devices; be *robust*, *fault tolerant* and *scalable; adapt* to environmental changes; and *be usable by novice users* via understandable designed models.

In this section, we shall briefly present several different approaches that deal with the open research issues of the UbiComp paradigm. We have grouped them based on a set of broad research areas, which nevertheless covers the UbiComp domain: middleware, architecture, user tools, applications and integration; in this chapter we do not concern ourselves with hardware issues and communication / networking protocols.

Research efforts on *user tools* try to ease the process of development for end-users who have little or no programming experience. The work by Humble *et al*. (Humble 2003) for example, uses a "jigsaw puzzle" metaphor in the Graphical User Interface (GUI). Individual devices and sensors are represented by puzzle piece-shaped icons that the user "snaps" together to build an application. While this metaphor is comprehensible, the interactions are simplified to sequential execution of actions and reactions depending on local properties (e.g. sensor events), which limits the potential to express many of the user's ideas. Similar approaches can be seen in the work of Truong et al (Truong 2004) that provides a pseudo-natural language interface, using a fridge magnet metaphor, and also in the browser approach of Speakeasy (Edwards 2002), where components are connected using a visual editor based on file-system browsers.

In (Jacquet 2005) is presented a clear conceptual model for ambient computing systems together with an architecture that introduces a high-level mechanism to abstract context and allows the rapid construction of ambient computing applications. The model uses a well-defined vocabulary and tries to map the physical and virtual

world to elementary component objects, which can be interconnected to form applications. The architecture however, limits the real world's representation to sets of sensors. This restricts the model's scope, while components loose their autonomy.

Other research efforts are emphasizing on the design of ubiquitous computing *architectures*. Project "Smart-Its" (Holmquist 2004) aims at developing small devices, which, when attached to objects, enable their association based on the concept of "context proximity". Objects are usually ordinary ones such as cups, tables, chairs etc., equipped with various sensors, as well as with a wireless communication module such as RF or Bluetooth. While a single Smart-It is able to perceive context information from its integrated sensors, a federation of ad hoc connected Smart-Its can gain collective awareness by sharing this information. However, the "augmentation" of physical objects is not related in any way with their "nature", thus the objects ends up to be just physical containers of the computational modules they host.

Project 2WEAR (Lalis et al. 2007, in this book) explores the concept of a personal system that is formed by putting together computing elements in an ad-hoc fashion using short-range radio. These elements may be embedded into wearable objects, or have the form of more conventional portable computers and mobile phones or are stationary elements that constitute a technological infrastructure. Interactions among objects are based on services, while users can build configurations flexibly using physical proximity as an interaction selection mechanism.

Representative approaches that focus on the development of *middleware* systems are BASE and Proem. BASE (Becker 2003; Weis 2006) presents a micro-kernel based middleware that is structured in multiple components that can be dynamically extended to interact with different existing middleware solutions and different communication technologies. While these approaches provide support for heterogeneity and a uniform abstraction of services the application programming interface requires specific programming capabilities (e.g., proxies as API) to building applications. Proem (Kortuem 2001) is a p2p platform supporting the application developer in creating and deploying applications. The objects managed by Proem are mainly electronic devices, such as PDAs and mobiles, and are abstracted as entities. Connectivity between entities is determined by proximity, while connected entities can form communities. Proem develops communication protocols that define the syntax and semantics of messages exchanged between peers, as well as an application environment including tools, APIs and logical structures. However, Proem does not consider multihop mobile ad hoc networks, while proximity poses severe limitations in the formation of UbiComp applications.

A recent research effort, ZUMA (Baker 2006) describes a platform based on a set of clean abstractions for users, content, and devices. The platform enables configuration and organization of content and networked heterogeneous devices in a smart-home environment. The main goals of this work is to achieve interconnection between incompatible devices, resolve conflicting personalization issues according to the users' properties, permissions and preferences, and achieve uniform control and manipulation of the environment. ZUMA employs the notion of multi-user optimal experience and attempts to achieve optimization by migrating applications to different environments. However, the information the system acquires by monitoring the environment via devices and sensor networks is combined with fixed knowledge e.g. regarding a person's permissions or preferences and the produced behaviour is in fact

predetermined actions based on simple rules. This results in the system operating without taking into consideration possible current user's feedback, while the user model for building pervasive applications is not very clear.

Finally, there are several approaches trying to establish some kind of *integrated*, preinstalled infrastructure in a physical area, e.g. a room or building, often called an intelligent environment (IE), in which the user and his/her mobile devices are integrated on-the-fly when entering the area. A representative research effort is taken by project Aura (Garlan 2002), which aims to "minimize distractions on user's attention, creating an integrated environment that adapts to the user's context and needs". Aura's goal is to provide each user with an invisible halo of computing and information services that persists regardless of location.

Project iROS (Johanson 2002), considers physically bounded spaces such as offices and meeting rooms that together provide low-level functionality. The system is modeled as an ensemble of entities that interact with each other using message passing. However, iROS does not provide explicit support for application development and management; instead, it relies on service synchronization using an event heap.

Last but not least, project "Ambient Agoras" (Streitz et al. 2005; Streitz et al. 2007, in this book) aims at providing situated services, place-relevant information, and feeling of the place ("genius loci") to the users, so that they feel at home in the office, by using mobile and embedded information technology. "Ambient Agoras" aims at turning every place into a social marketplace (= "agora") of ideas and information where people can interact and communicate.

All these research efforts, together with various research initiatives (i.e. DC 2006) have given us a glimpse of what the UbiComp-enabled future might perhaps bring. As Weiser noted in his seminal paper, we don't really know what's coming (Weiser 1993): "*Neither an explication of the principles of ubiquitous computing nor a list of the technologies involved really gives a sense of what it would be like to live in a world full of invisible widgets. To extrapolate from today's rudimentary fragments of embodied virtuality resembles an attempt to predict the publication of Finnegan's Wake after just having invented writing on clay tablets. Nevertheless the effort is probably worthwhile."*

3 A New Approach for Shaping UbiComp Environments

The ways that we can use an ordinary object are a direct consequence of the anticipated uses that object designers "embed" into the object's physical properties. This association is in fact bi-directional: the objects have been designed to be suitable for certain tasks, but it is also their physical properties that constrain the tasks people use them for. According to D. Norman (Norman 1988) affordances "refer to the perceived and actual properties of the thing, primarily those fundamental properties that determine just how the thing could possibly be used". We therefore say that an object may "afford" some sorts of action, and when it does so, this results in a set of natural or "easy" relations.

Due to their "digital self", AmI objects can now publicize their abilities in the digital space. These include properties (what the object is), capabilities (what the object knows to do) and services (what the object can offer to others). At the same time, they

acquire extra capabilities, which during the formation of UbiComp applications, can be combined with capabilities of other AmI objects or adapt to the context of operation. Thus, AmI objects offer two new affordances to their users:

- Composeability: artifacts can be used as building blocks of larger and more complex systems. Composeability is perceived by users through the presentation -via the object's digital self- of the object's connectable capabilities, and thus provide users the possibility to achieve connections and compose applications of two or more objects.
- Changeability: artifacts that possess or have access to digital storage can change the digital services they offer. In other words, the tangible object can be partially disassociated from the artifact's digital services, because the latter result from the execution of stored programs.

Both these affordances are a result of the ability to produce descriptions of properties, abilities and services, which carry information about the artifact in the digital space. This ability improves object / service independence, as an AmI object that acts as a service consumer may seek a service producer based only on a service description. For example, consider the analogy of someone wanting to drive a nail and asking not for the hammer, but for any object that could offer a hammering service (could be a large flat stone). In order to be consistent with the physical world, functional *autonomy* of AmI objects must also be preserved; thus, they must be capable to function without any dependencies from other AmI objects or infrastructure.

Newman states that "Systems should inherently support the ability of users to assemble available resources to accomplish their tasks. ….there will always be particular combinations of functionality for which no application has been expressly written" (Newman 2002). By considering AmI objects as resources, Newman's statement points out that the possible combinations of AmI objects cannot be foreseen, since people –who create and use these objects- are inherently unpredictable.

Instead of trying to predetermine a range of likely UbiComp applications, what seems more familiar to people's way of acting is to make AmI objects *composeable,* that is, make them capable of being easily composed as building blocks into larger and more complex systems. In this way, designing and implementing UbiComp applications could become an apprehensible procedure that even non experts can carry out provided that they are supported by proper end-user tools. Moreover, this approach is also more economically viable, as new applications can be created by sharing or reusing existing AmI objects, thus minimizing the need for acquiring new components.

For the AmI vision to succeed, nobody should be excluded from using UbiComp technology or accessing UbiComp applications or system services. The designers / users dichotomy appears again, only this time it is easier for users to act as designers themselves. People actively participate in shaping UbiComp environments, so the provision of models and metaphors, cognitive or semantic, is necessary to make sure that these environments are coherent and understandable.

The UbiComp paradigm introduces several challenges for people. At first, users will have to update their existing task models, as they will no longer interact with an ordinary object but with a computationally enabled AmI object. Then, people will have to form new models about the everyday objects they use. Finally, as the human-computer interface integrates with the physical world, the prevailing human-computer

interaction paradigms (such as the direct manipulation paradigm) will be infused by human-object interaction metaphors, thus becoming more natural.

A similarly large range of challenges are brought forward for the untrained designers. The ones relating to technology have been already discussed. These are complemented by the need to produce designs of controlled obtrusion, handling the new affordances of objects and at the same time minimize disturbance of existing models.

Thus, we assert that on the way to realizing the AmI vision, together with the realization of ubiquitous computing technology, a conceptual framework that will bridge the gap between system design and use is necessary. This framework must be shared both by developers and end users so that the latter are enabled to actively shape the ubiquitous computing environments they live in, without them being hindered by design limitations. Moreover, the visibility of the functionality of the UbiComp system must be transparent and adapting to people; people have to remain informed and aware, in order to build trust on the system. The framework must be complemented with the technological platform and tools that will facilitate users' active participation in the adoption, formation and use of UbiComp applications.

3.1 The Gadgetware Architectural Style (GAS)

We have designed GAS (the Gadgetware Architectural Style), as a conceptual and technological framework for describing and manipulating ubiquitous computing applications (Kameas 2003). It consists of a set of architecture descriptions (syntactic domain) and a set of guidelines for their interpretation (semantic domain). GAS extends component-based architectures to the realm of tangible objects and combines a software architectural style with guidelines on how to physically design and manipulate artefacts (we could call it "a style for tangible objects") (Kameas 2005).

For the end-user, this model can serve as a high level task interface; for the developer, it can serve as a domain model and a methodology. In both cases, it can be used as a communication medium, which people can understand, and by using it they can manipulate the "disappearing computers" within their environment.

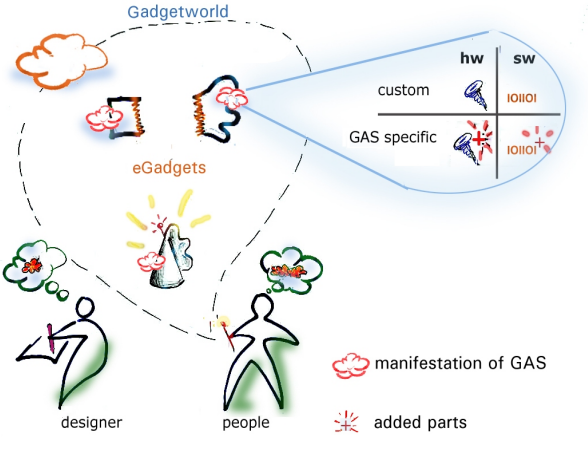

Fig. 1. GAS, an Architectural style that acts as a common referent between people, designers, and artifacts

In our approach, we view the process whereby people configure and use complex collections of interacting artifacts, as having much in common with the process where system builders design software systems out of components. We consider the everyday environment to contain a multitude of artifacts, which people combine and recombine in ad-hoc, dynamic ways. As a consequence, artifacts are treated as reusable "components" of a dynamically changing physical or digital environment; people are given "things" with which to make "new things", rather than only being supplied with fixed and un-changeable tools. The behavior of these "new things" (which are ubiquitous computing applications) is neither static, nor random, as it is guided by how applications are to be used (e-Gadgets 2006).

The underlying hypothesis is that even if an individual artifact has limited functionality, it can have more advanced behavior when grouped with others. Composeability can give rise to new collective functionality as a result of unprecedented but well-defined interactions among artifacts. Then the aim is to look at how collections of artifacts can be configured to work together in order to provide behavior or functionality that exceeds the sum of their parts.

GAS defines a vocabulary of entities and functions, a set of configuration rules (for interactively establishing associations between artifacts), and a technical infrastructure, the GAS-OS middleware (algorithms, protocols, interfaces, etc). It is conceived so as to be compatible with the mental models of ubiquitous applications that are maintained by artifact manufacturers (i.e. in the form of design guidelines and APIs) or people acting as application composers (i.e. in the form of configuration rules and constraints for composing artifact societies); it can also be represented in the collaboration logic of artifacts, in the form of communication protocol semantics and algorithms (Figure 1.).

GAS aims to serve as a consistent conceptual and technical referent among artefact designers and application designers, by allowing the former to provide the building blocks and rules for the latter to compose functional applications. It also serves as conveyor of design semantics from design experts to application designers. Design options and compositional constraints can be embedded in artifacts in the form of configuration rules which guide and inform the composition and use of applications. GAS plays the role of a vehicle that enables people to become creative shapers of their environment - rather than passive consumers - by enabling them to create new and emerging functionalities by composing pre-fabricated artifacts.

3.2 Harnessing the Challenges of UbiComp Environments

In this section, we shall discuss the extent to which GAS-compatible UbiComp applications meet the UbiComp system design challenges.

Heterogeneity results from the fact that components of a UbiComp system have to be implemented using different hardware (e.g. computational boards, sensors/actuators, etc.), operating systems, programming languages and have to communicate with other nodes through different networks (e.g. 802.11, Bluetooth, Ethernet, infrared, etc.). GAS confronts the heterogeneity issues with the help of GAS middleware (GAS-OS) running on a Java virtual machine. The use of Java as the underlying platform for the middleware facilitates its deployment on a wide range of devices from mobile phones and PDAs to specialized Java processors, hiding the underlying computational units

and operating systems. GAS-OS also implements a driver-based mechanism to mask the heterogeneity of sensors and actuators, while the development of a modular JXTA-like communication module makes GAS-OS capable of working over the most known standards, such as 802.11, Bluetooth, infrared, and Ethernet, while easily extendible to support other communication technologies as well. Thus the combination of the Java platform and the GAS-OS middleware, solve to a certain extent the heterogeneity issues that may arise when different GAS enabled artifacts need to collaborate.

The need to support mobility is very frequent in UbiComp environments, as many of the nodes that constitute an application may be portable or even wearable devices. As a consequence the structure of the distributed system is changing dynamically which may also lead to service unavailability in cases where a node offering a certain service gets out of the application's range. GAS-OS faces this challenge by using short or long range wireless networking protocols depending on the application's requirements. Furthermore using hybrid routing protocols reduces the probability to loose a moving node. Even in the case where a service eventually becomes unavailable GAS-OS offers service replacement functionality discovering nodes offering the same service in order to replace the missing one.

Designing a UbiComp system to be composeable allows the nodes of the system (e.g. artifacts) to be used as building blocks of larger and more complex systems. GAS builds on the foundations of established software development approaches such as object oriented design and component frameworks, and extends these concepts by exposing them to the end-user. To achieve this, GAS defines the Plug/Synapse model (Figure 2) which provides a high-level abstraction of the component interfaces and the composition procedure.

Using the Plug/Synapse model GAS also achieves context awareness. Context is collected independently by each artifact via its sensors, is semantically interpreted using its stored ontology and manifested via its plugs. Synapses are the means to propagate each artifact's context to other components, resulting in context aware applications.

A further challenge in UbiComp systems is to preserve each object's autonomy, according to which artifacts must function independently of the existence of other artifacts. Taking into consideration the ad-hoc nature of UbiComp environments and the fact that service unavailability is a frequent phenomenon rather than an exception, object autonomy seems to be a prominent factor for keeping an application functional. Furthermore, object autonomy makes applications more flexible and portable as their operation does not depend on any kind of infrastructure. Preserving object autonomy implies that even resource constraint devices would have to be self contained. GAS-OS copes with this challenge by adapting ideas from the microkernel design where only minimal functionality is located in the kernel, while extra services can be added as plug-ins. This way all objects, even resource constraint ones, are capable of executing the kernel which allows them to be autonomous and participate in applications.

Preserving object autonomy is also a way to make UbiComp applications robust and fault tolerant as the failure of one node does not necessarily mean that the whole system fails. Other ways GAS uses to increase the system's credibility is the provision of reliable p2p communication protocols, the replacement of services in case of unavailability, the provision of routing protocols to increase node accessibility, etc.

A very important challenge in UbiComp systems is scalability. Systems must be scalable in terms of numbers of users and services, volume data stored and manipulated, rates of processing, numbers of nodes and sizes of networks and storage devices. Scalability means not just the ability to operate, but to operate efficiently and with the adequate quality of service over the given range of configurations. GAS-OS is designed to support a large number of nodes by encompassing a lightweight communication module supporting p2p asynchronous communication. In order to avoid large messages and the resulting traffic congestion in the network, XML-based messages are used to wrap the information required for each protocol. Furthermore, polling techniques that usually cause message flooding and bandwidth reduction are avoided. Routing is also designed as a hybrid protocol that combines a proactive and a reactive part, trying to minimize the sum of their respective overheads.

Up to now, the ways that an object could be used and the tasks it could participate in have usually been determined by it's physical affordances, that are, in turn, determined by its shape, material, and physical properties. GAS enabled artifacts overcome this limitation by producing descriptions of their properties, abilities and services in the digital space using ontologies, thus becoming able to improve their functionality by participating in compositions, learning from usage, thus becoming adaptive.

3.3 Plug/Synapse: The GAS Conceptual Framework

The basic concepts encapsulated in the Plug/Synapse model are:

eGadget: eGadgets are everyday physical objects enhanced with sensing, acting, processing and communication abilities. eGadgets can be regarded as artifacts that can be used as building blocks to form GadgetWorlds.

Plugs: From a user's perspective, they provide for the possibility of connectivity, as they make visible the artifacts' properties, capabilities and services to people and to other artifacts; they are implemented as software classes.

Synapses: They are associations between two compatible plugs, which make use of value mappings; they are implemented using a message-oriented set of protocols.

GadgetWorld: A GadgetWorld is a dynamic distinguishable, functional configuration of associated eGadgets, which communicate and / or collaborate in order to realize a collective function. GadgetWorlds are formed purposefully by an actor (user or other).

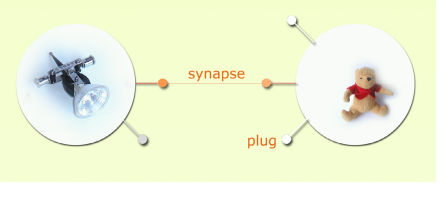

Fig. 2. The Plug Synapse model: The artifacts' capabilities (Plugs) can be inter-associated with invisible links (Synapses) to form ubiquitous computing applications

To achieve the desired collective functionality, one forms synapses by associating compatible plugs, thus composing applications using artifacts as components. Two levels of plug compatibility exist: direction and data type compatibility. According to direction compatibility output or I/O plugs can only be connected to input or I/O plugs. According to data type compatibility, plugs must have the same data type to be connected via a synapse. However, this is a restriction that can be bypassed using value mappings in a synapse (Figure 2). No other limitation exists in making a synapse. Although this may mean that seemingly meaningless synapses are allowed, it has the advantage of letting the user create associations and cause the emergence of new behaviours that the artifact manufacturer may have never thought of. Meaningless synapses could be regarded as analogous to "logical errors" in a program (i.e. a program may be compiled correctly but does not manifest the desired by the programmer behavior).

3.4 Methodology

According to AmI vision, people live in an environment populated with artifacts; they have a certain need or task, which they think can be met or carried out by (using) a combination of services and capabilities. Then, in this approach, they search for artifacts offering these services and capabilities as plugs; they select the most appropriate ones and combine the respective plugs into functioning synapses; if necessary, they manually adapt or optimize the collective functionality. Note that we are referring to the selection of plugs, not artifacts, thus preserving physical independence; this selection is task-based.

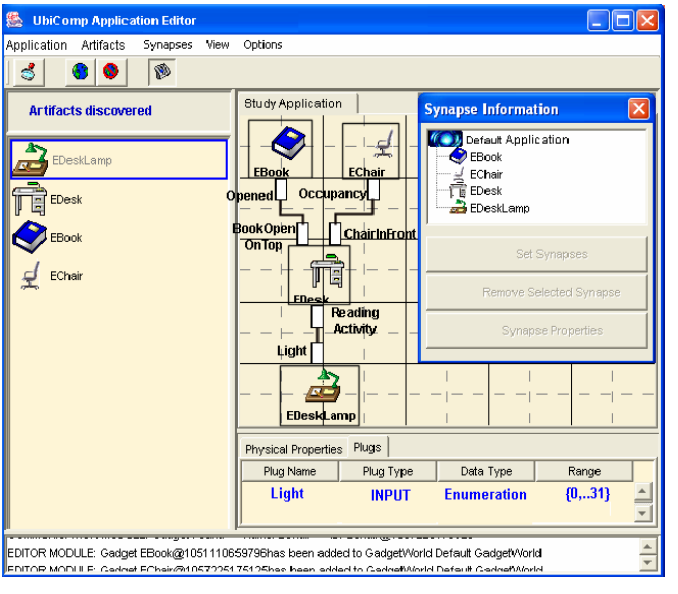

Fig. 3. Combined artifacts in the UbiComp application editor

The use of high-level abstractions, for expressing artifact associations, allows the flexible configuration and reconfiguration of UbiComp applications. It only requires that they are able to communicate. Moreover, they have to run the GAS-OS middleware in order to "comprehend" each-other, so that people can access their services, properties and capabilities in a uniform way. To support people in this process and to hide the complexity of artifact interactions, appropriate tools were developed that implement the conceptual framework (Mavrommati 2004) (Figures 3 and 4). People in that way do not need to be engaged in any type of formal "programming" in order to achieve the desired functions.

Because synapses are formed by end-users, who are not programmers, they can only be debugged by "trial and error" approach. In this case however, what, from a programmer's standpoint, may seem as erroneous behaviour, or inappropriate synapse "programming", from the end user's standpoint and in certain occasions, may simply cause the manifestation of unexpected, unprecedented, but still interesting and desirable system behaviour.

4 Building a Home Application

These concepts can be better illustrated if we consider the Study application example, adopted from (Drossos 2006), which we'll follow throughout this chapter. Let's take a look at the life of Pat, a 27-year old single woman, who lives in a small apartment near the city centre and studies Spanish literature at the Open University. A few days ago she passed by a store, where she saw an advertisement about these new augmented artifacts. Pat decided to enter. Half an hour later she had given herself a very unusual present: a few furniture pieces and other devices that would turn her apartment into a smart one! On the next day, she was anxiously waiting for the delivery of an eDesk (it could sense objects on top, proximity of a chair), an eChair (it could tell whether someone was sitting on it), a couple of eLamps (one could remotely turn them on and off), and some eBook tags (they could be attached to a book, tell whether a book is open or closed). Pat had asked the seller to pre-configure some of the artifacts, so that she could create a smart studying corner in her living room. Her idea was simple: when she sat on the chair and she would draw it near the desk and then open a book on it, then the study lamp would be switched on automatically. If she would close the book or stand up, then the light would go off.

The behavior requested by Pat requires the combined operation of the following set of artifacts: eDesk, eChair, eDeskLamp and eBook. The properties and plugs of these artifacts are shown in Table 1 and are manifested to Pat via the UbiComp Application Editor tool (Mavrommati 2004), an end-user tool that acts as the mediator between the plug/synapse conceptual model and the actual system. Using this tool Pat can combine the most appropriate plugs into functioning synapses as shown in Figure 3.

In the case of the synapse between eDesk.ReadingActivity and eDeskLamp.Light plugs, a data type compatibility issue arises. To make the synapse work, Pat can use the UbiComp Editor to define mappings that will make the two plugs collaborate, as shown in Figure 4.

Table 1. Analyzing the UbiComp Application

Artifact	Properties	Plugs	Functional Schemas
eChair	Sensing chair occupancy capability (C_1) Transmitting object type capability (C_2)	Occupancy: {OUT | Boolean}	eChair.C_1 ← read(pressure-sensor) eChair.C2 is an attribute Occupancy ← {eChair.C_1, eChair.C2}
eBook	Sensing open/close capability (C_1) Transmitting object type capability (C_2)	Opened {OUT | Boolean}	eBook.C1 ← read(bend-sensor) eBook.C2 is an attribute Opened ← {eBook.C_1, eBook.C2}
eDesk	Sensing objects on top capability (C_1) Sensing proximity of objects capability (C_2)	BookOpenOnTop: {IN | Boolean} ChairInFront: {IN | Boolean} ReadingActivity: {OUT | Boolean}	eDesk.C1 ← read(RFID-sensor) eDesk.C2 ← read(proximity-sensor) IF eDesk.C1 = eBook.C2 AND eBook.C1 = TRUE THEN BookOpenOnTop ← TRUE ELSE BookOpenOnTop ← FALSE IF eDesk.C2 = TRUE AND eChair.C_1 = TRUE THEN ChairInFront ← TRUE ELSE ChairInFront ← FALSE IF BookOpenOnTop = TRUE AND ChairInFront = TRUE THEN ReadingActivity ← TRUE ELSE ReadingActivity ← FALSE
eDesk-Lamp	Light service (S_1)	Light: {IN | Enumeration}	IF eDesk.ReadingActivity THEN S1(on) ELSE S1(off)

The definition of the functional schemas of the artifacts, that is the internal logic that governs the behaviour of each artifact either when its state changes or when a synapse is activated are predefined by the artifact developer (for our example, they are shown in Table I). Rules that require identification of the remote artifact can be specified using the property schema information, which is available in the representation of each of the two artifacts that participate in a synapse.

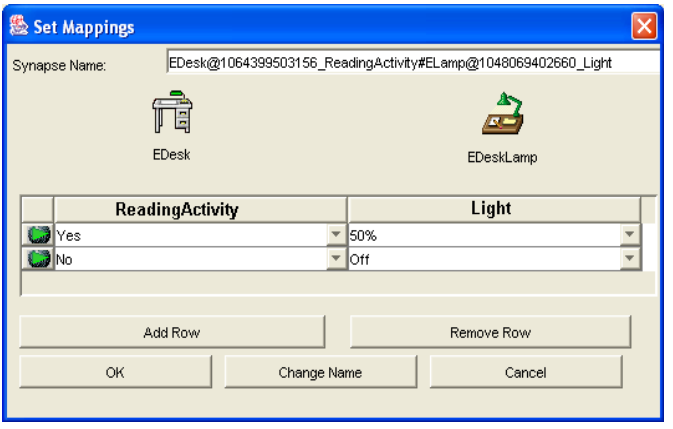

Fig. 4. Setting mappings between the eDesk.ReadingActivity and eDeskLamp.Light plugs

The eBook, eChair and eDesk comprise an artifact composition (GadgetWorld) whose composite property is manifested via the ReadingActivity plug. This plug allows the participation of this composition, in a way similar to any artifact, in other compositions. In this way, there's no conceptual upper limit in the structural complexity of GAS enabled UbiComp applications. Any GadgetWorld can be edited to extend the functionality of the application. For example, consider that Pat also buys an eClock and wants to use it as a 2 hour reading notification. The eClock owns an Alarm plug that when activated, via a synapse, counts the number of hours (which can be configured) and then rings the alarm. To implement her idea, what Pat has to do is to use the UbiComp Application editor to create a synapse between the ReadingActivity plug of the eDesk and the Alarm plug of the eClock and specify the number of hours in the Properties dialog box of the eClock.

5 GAS-OS: The GAS Middleware

To cope with heterogeneity and provide a uniform abstraction of artifact services and capabilities we have introduced the GAS-OS middleware that abstracts the underlying data communications and sensor/actuator access components of each part of a distributed system, so that a UbiComp application appears as an integrated system. GAS-OS follows the Message-Oriented Middleware (MOM) approach, by providing non-blocking message passing and queuing services. Furthermore, to handle the need to adapt to a broad range of devices, we have adapted ideas from micro-kernel design (Tanenbaum 1991) where only minimal functionality is located in the kernel, while extra services can be added as plug-ins.

We assume that no specific networking infrastructure exists, thus ad-hoc networks are formed. The physical layer networking protocols used are highly heterogeneous ranging from infrared communication over radio links to wired connections. Since every node serves both as a client and as a server (devices can either provide or request services at the same time), communication between artifacts can be considered as Peer-to-Peer (P2P) (Schollmeier 2001).

The prototype of GAS-OS is implemented in Java but relies only on features available in the Java personal edition, compatible with the Sun J2ME Personal Profile. This allows the deployment on a wide range of devices from mobile phones and PDAs to specialized Java processors. The proliferation of systems besides classical computers capable of executing Java, make it a suitable underlying layer providing a uniform abstraction for our middleware.

The use of Java as the underlying platform of the middleware decouples GAS-OS from typical operations like memory management, networking, etc. Furthermore, it facilitates the deployment on a wide range of devices from mobile phones and PDAs to specialized Java processors.

The combination of the Java platform and the GAS-OS middleware, hide the heterogeneity of the underlying artifacts, sensors, networks etc. and provides the means to create large scale systems based on simple building blocks.

5.1 Design

The key idea behind GAS-OS is the uniform abstraction of artifact services and capabilities via the plug/synapse high-level programming model that abstracts the underlying data communications and access components of each part of a distributed system (Figure 5). It isolates clients from back-end processes and decouples the application from the network layer. Messaging and queuing allow nodes to communicate across a network without being linked by a private, dedicated, logical connection. The clients and servers can run at different times. Every node communicates by putting messages on queues and by getting messages from queues.

Our approach is inspired by micro-kernels. Only minimal functionality is located in the kernel, while extra services can be added as plug-ins. One issue in the design

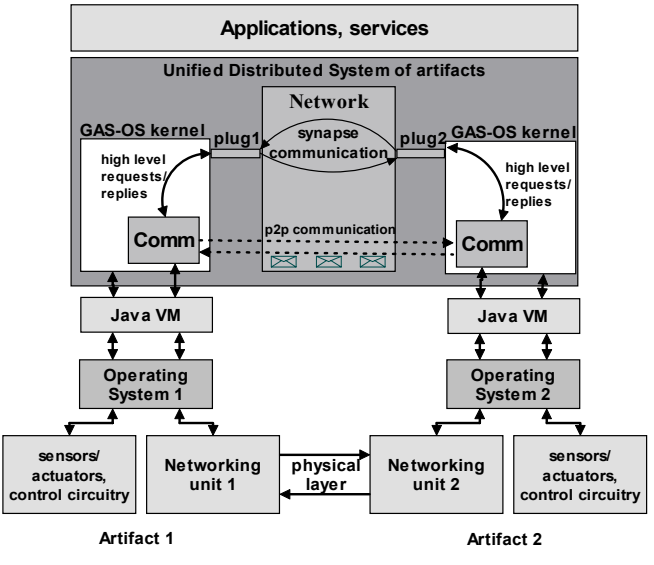

Fig. 5. GAS-OS unifying a distributed system of artifacts

of GAS-OS was to decide on the set of minimal functionalities. The policy that was adopted was to maintain the autonomous nature of artifacts but at the same time make even the more resource constraint ones capable of participating to ubiquitous applications. As a consequence, the kernel of GAS-OS accepts and dispatches messages and manages local hardware resources and the Plug/synapse interoperability protocols.

The GAS-OS kernel implements plugs in order to manifest the artifact's services and capabilities and can initiate or participate in a synapsing process using the Plug/synapse interoperability protocol. The communication module, translates the high-level requests/replies into messages and using low-level peer-to-peer networking protocols, dispatches them to the corresponding remote service or device capability (Figure 5). The kernel is also capable of handling service and resource discovery messages in order to facilitate the formation of synapses.

5.2 Architecture

The outline of the GAS-OS architecture is shown in Figure 6 (adopted from (Drossos 2006), where it is presented in more detail). The GAS-OS kernel is designed to accept and dispatch messages, manage local hardware resources (sensors/actuators), and implement the plug/synapse interaction mechanism. The kernel is also capable of managing service and artifact discovery messages in order to facilitate the formation of the proper synapses.

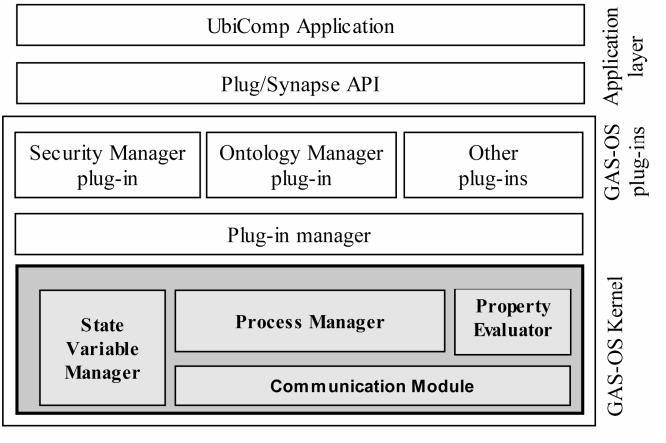

Fig. 6. GAS-OS modular architecture

The GAS-OS kernel encompasses a P2P Communication Module, a Process Manager, a State Variable Manager, and a Property Evaluator module as shown in Figure 6. The P2P Communication Module is responsible for application-level communication between the various GAS-OS nodes. This module translates the high-level requests/replies into messages and by using low-level networking protocols it dispatches them to the corresponding remote peers. The Process Manager is the coordinator module of GAS-OS. Some of its most important tasks are to manage the processing policies, to

accept and serve requests set by the other modules of the kernel or to initiate reactions in collaboration with other modules, tasks which collectively serve the realization of the Plug/Synapse model. Furthermore, it is responsible for handling the memory resources of an artifact and caching information of other artifacts to improve communication performance when service discovery is required. The State Variable Manager handles the runtime storage of artifact's state variable values, reflecting both the hardware environment (sensors/actuators) at each particular moment (primitive properties), and properties that are evaluated based on sensory data and P2P communicated data (composite properties). The Property Evaluator is responsible for the evaluation of artifact's composite properties according to its Functional Schema. In its typical form the Property Evaluator is based on a set of rules that govern artifact transition from one state to another. The rule management can be separated from the evaluation logic by using a high-level rule language and a mechanism that translates high-level rule specifications to XML that can be exploited then by the evaluation logic.

The adoption of a layered modular architecture allows the replacement of a module without affecting the functionality of the rest provided that the APIs between them remain consistent. This principle holds for the different layers of the architecture as well as within each layer. The modular design of GAS-OS, for example, allows the integration of up-to-date algorithms and protocols in the form of plug-in modules. Extending the functionality of the GAS-OS kernel can be achieved through plug-ins, which can be easily incorporated to an artifact running GAS-OS, via the plug-in manager.

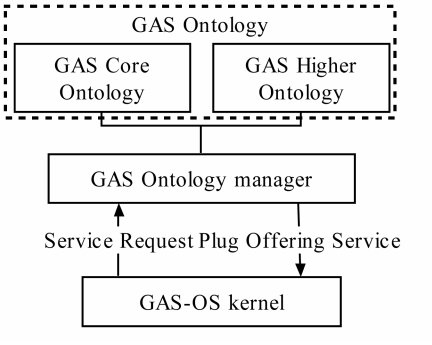

Fig. 7. The GAS Ontology manager

GAS Ontology (Christopoulou 2005) describes the semantics of the basic terms of our model for UbiComp applications and their interrelations. It contains a service classification, since the AmI objects offer various services and the demand for a semantic service discovery mechanism in UbiComp applications is evident. Due to the facts that AmI objects acquire different knowledge and may have limited capabilities, the GAS Ontology was divided into two layers: the GAS Core Ontology (GAS-CO) and the GAS Higher Ontology (GAS-HO). The GAS-CO is fairly small and provides AmI objects with the necessary common language that they need in order to describe their acquired knowledge represented by the GAS-HO. In that way, high-level descriptions of services and resources independent of the context of a specific application are possible,

facilitating the exchange of information between heterogeneous artifacts as well as the discovery of services.

Figure 7 demonstrates the interaction among the ontology manager and the GAS-OS kernel. An important feature of the ontology manager is that it adds a level of abstraction between GAS-OS and the GAS ontology, because only the ontology manager can manipulate the ontology; GAS-OS can simply query this module for information stored into the ontology without having any knowledge about the ontology language or its structure. GAS-OS receives from the ontology manager the necessary knowledge stored in an AmI object's ontology relevant to its services, in order to implement a service discovery mechanism. The GAS Ontology manager using this mechanism and the service classification can identify AmI objects that offer similar semantically services and propose objects that can replace damaged ones.

The security manager plug-in on the other hand, when developed, will be responsible for realizing the security policies of each artifact. These policies will be encoded as rules in the ontology, thus becoming directly available to the Process Manager. The security manager will mediate information exchange via synapses in order to ensure that security policies are respected.

6 The Evolution of GAS

GAS is an architectural style which represents a worldview of everyday living that directly descends from the AmI vision. In a world where all objects in our living spaces are built according to this style, a whole new way of interacting with our environment can emerge. The success of this vision depends both on technological advancements that need to be made, as well as on future research on GAS-like components, such as the conceptual framework, the middleware, the ontology, and the user editing tools.

Advancements that have to be achieved involve first of all hardware miniaturization. If we want to consider computers "disappearing", then it is crucial that hardware units like processors, sensors, networking boards etc. become small enough so that their actual embedding in all kinds of objects, no matter their size or shape, is possible. Furthermore, low power consumption is essential especially for mobile and wearable objects as it is closely related to longer battery life and object autonomy. This can also be promoted by the use of low power networking technologies like short range RF or low-frequency RF, without sacrificing the quality of communication. Advancements must also be made regarding sensors not only with respect to their size but also to their precision, fault tolerance etc. Finally, since the hardware part of the eGadget is the most expensive one, efforts have to be made to reduce the overall cost. This will make these "augmented objects" affordable for everyday people, thus achieve their intended purpose, to be used widely in everyday applications.

In addition, a number of enhancements in software must also be taken into consideration. Software systems must target more on the human's perspective rather than technology itself. To achieve this, software must become adaptive to people's needs making UbiComp applications as usable by people as possible, reducing the required time and effort in learning how to interact with the system. User oriented tools must

also be developed to facilitate people's intervention in their living environments, while keeping it simple to actually live in such environments.

Along these lines the design and development of a security/privacy module on top of the middleware (e.g. as a plug-in) would definitely extend the functionality of GAS-OS to realizing the security policies of each artifact as well as handle privacy issues in applications, which are proved to be of utmost importance. Especially, when envisaging a world composed of UbiComp applications executing all together, the provision of ways to prohibit violations and undesirable interactions among them is prominent. This module must also implement mechanisms to set digital boundaries to the otherwise unlimited access in computational environments, in order to ensure that as private spaces are limited by "walls" in the physical world, they will also be restricted via privacy algorithms and techniques in the digital world.

Further research is also required for the development of a resource management module, capable of dealing with physical resources scarcities by providing resource mapping and mediation services. This module has to keep track of the available resources and arbitrate among conflicting requests for those resources. Local management of those resources is already undertaken by the local operating system of each node, thus the role of the resource management module is to globally manage resources at the application level. Examples are allowing eGadgets lending their resources under certain circumstances (e.g. certain time interval, certain QoS, etc.) to other eGadgets which do not have the adequate resources to execute a task with the required performance, searching for eGadgets with adequate resources before assigning a task, etc..

Achieving an acceptable level of quality of services is also very important in UbiComp applications, thus a separate module ensuring that the provided services will meet certain requirements is needed. QoS affects all layers in a UbiComp application, from the application layer ensuring real-time system responses, stability, constant service provision for a certain time interval, down to the networking layer ensuring reliable communication, high data transmission rates, etc.. Dynamically supporting QoS also contributes to adaptation, as the system dynamically changes its parameters to maintain the required QoS level. QoS parameters vary depending on the application, thus the QoS module should be able to receive requirements, probably by the developer of the application, while automatic determination has to be investigated.

Furthermore, as ecologies of collaborating artifacts are dynamically reconfigured aiming at the accomplishment of tasks, their formation heavily depends not only on space and time but also on the context of previous local interactions, previous configured teams, successfully achieved goals or possible failures. This means that in order to initially create, manage, communicate with, and reason about, such kinds of emergent ecologies, we need to model and embed to these entities social memory, enhanced context memory, models of self and shared experiences. One way to achieve this is to design and implement evolving multidimensional ontologies that will include both nonfunctional descriptions, goals, rules and constraints of application, as well as aspects of dynamic behaviour and interactions. A core ontology will be open and universally available and accessible; however, during the ecology lifetime the core ontology is evolved into higher goal, application and context specific one. Hence, ontologies describing specific application domains can be proprietary.

In order to make UbiComp applications as usable by people as possible, the use of intelligent agents is an option that could possibly reduce people's time and effort in learning how to interact with the system. The idea is to have software agents monitoring and learning how people actually want to use their augmented environments. Based on monitored data, agents are capable, using either rule-based or even fuzzy logic, of inferring or even predicting the ways people expect their environment to behave.

Finally, the improvement of the existing UbiComp application editing tools aimed at end users as well as the development of new ones is a step forward towards the closest and easiest people's involvement into shaping their living spaces. The UbiComp Application Editor is a prototype that realizes the people's requirements from GAS using the plug/synapse model (Figure 8). The experiences reported after expert and user trials suggest that an architectural approach where users act as composers of predefined components is worthwhile (Markopoulos 2004). However, these sessions have also pointed out that further improvement is necessary in the design and interaction of such tools. Providing for adaptive interfaces, to cater for a variety of user profiles as well as augmented reality and speech interfaces, pose a number of challenges on the design of editing tools

Fig. 8. Test subjects using the GAS Application Editor, during a user trial session

Additional tools are also required to harness the ubiquity and the logic governing this new type of environments. Such tools are addressed more to the developer or the advanced user rather to an everyday end-user. The purpose of these tools is to configure certain aspects of the system's behavior that need user intervention. Decision making is one such aspect according to which the developer or advanced user may want to dynamically define or change the rules determining an individual eGadget's or even an application's behavior in a high level manner. Dynamically defining QoS parameters for an application is another aspect that can be defined or altered using tools.

All in all, there are many achievements waiting to be realized and paths to be explored in order to make computers ubiquitous and well accepted by people in their

everyday living. As Weiser noted in his article "The computer for the 21st century" (Weiser 1991), "There is more information available at our fingertips during a walk in the woods than in any computer system, yet people find walk among trees relaxing and computers frustrating. Machines that fit the human environment instead of forcing humans to enter theirs, will make using a computer as refreshing as taking a walk in the woods."

Acknowledgements

Part of the research described in this chapter was conducted in the "extrovert Gadgets" project (funded under project number IST-2000-25240 in the context of the European funded IST-FET "Disappearing Computer" initiative). The authors would like to thank fellow e-Gadgets researchers at CTI, Tyndall and University of Essex as well as the TU/e experts and all the experts involved in our research and studies.

References

Accord project website (2006): http://www.sics.se/accord/ (last accessed on 11/22/2006)

Baker, C.R., Markovsky, Y., van Greunen, J., Rabaey, J., Wawrzynek, J., Wolisz, A.: ZUMA: A Platform for Smart-Home Environments. In: Proceedings of the 2nd IET Conference on Intelligent Environments, Athens, Greece (2006)

Becker, C. et al.: BASE - A Micro-broker-based Middleware For Pervasive Computing. In: Proceedings of the 1st IEEE International Conference on Pervasive Computing and Communication (PerCom03), Fort Worth, USA (2003)

Lalis, S., Savidis, A., Karypidis, A., Gutknecht, J., Stephanides, C.: Towards Dynamic and Cooperative Multi-Device Personal Computing. In: Streitz, N., Kameas, A., Mavrommati, I. (eds.) The Disappearing Computer. LNCS, vol. 4500, Springer, Heidelberg (2007)

Christopoulou, E., Kameas, A.: GAS Ontology: an ontology for collaboration among ubiquitous computing devices. International Journal of Human – Computer Studies 62(5), 664–685 (2005)

Disappearing Computer initiative (2006): http://www.disappearing-computer.net (last accessed on 11/22/2006)

Drossos, N., Goumopoulos, C., Kameas, A.: A Conceptual Model and The Supporting Middleware For Composing Ubiquitous Computing Applications. Special Issue in the Journal of Ubiquitous Computing and Intelligence (JUCI), entitled Ubiquitous Intelligence in Real Worlds, American Scientific Publishers (ASP) 1(2), 1–13 (2006)

e-Gadgets project website (2006): http://www.extrovert-gadgets.net (last accessed on 11/22/2006)

Edwards, W.K., Newman, M.W., Sedivy, J., Smith, T., Izadi, S.: Challenge: Recombinant Computing and the Speakeasy Approach. In: Proceedings of the Eighth Annual International Conference on Mobile Computing and Networking (MobiCom 2002), pp. 279–286. ACM Press, New York (2002)

Garlan, D., Siewiorek, D.P., Smailagic, A., Steenkistie, P.: Project Aura: Toward Distraction-Free Pervasive Computing. IEEE Pervasive Computing Magazine 1(2), 22–31 (2002)

Holmquist, L.E., Gellersen, H.-W., Schmidt, A., Strohbach, M., Kortuem, G., Antifakos, S., Michahelles, F., Schiele, B., Beigl, M., Mazé, R.: Building Intelligent Environments with Smart-Its. IEEE Computer Graphics & Applications 24(1), 56–64 (2004)

Humble, J. et al.: Playing with the Bits: User-Configuration of Ubiquitous Domestic Environments. In: Dey, A.K., Schmidt, A., McCarthy, J.F. (eds.) UbiComp 2003. LNCS, vol. 2864, pp. 256–263. Springer, Heidelberg (2003)

ISTAG ISTAG in FP6: Working Group 1, IST Research Content, Final Report, available at http://www.cordis.lu/ist/istag.htm (last accessed on 11/22/2006)

Jacquet, C., Bourda, Y., Bellik, Y.: An Architecture for Ambient Computing. In: Proceedings of the 1st IEE International Workshop on Intelligent Environments, Colchester, UK, pp. 47–54 (2005)

Johanson, B., Fox, A., Winograd, T.: Experiences with Ubiquitous Computing Rooms. IEEE Pervasive Computing Magazine 1(2), 67–74 (2002)

Kameas, A. et al.: An Architecture that Treats Everyday Objects as Communicating Tangible Components. In: Proceedings of the 1st IEEE International Conference on Pervasive Computing and Communications (PerCom03), Fort Worth, USA, pp. 115–122. IEEE Computer Society Press, Los Alamitos (2003)

Kameas, A., Mavrommati, I.: Configuring the e-Gadgets. Communications of the ACM (CACM) 48(3), 69 (2005)

Kortuem, G., Schneider, J.: An Application Platform for Mobile Ad-hoc Networks. In: Abowd, G.D., Brumitt, B., Shafer, S. (eds.) Ubicomp 2001: Ubiquitous Computing. LNCS, vol. 2201, Springer, Heidelberg (2001)

Markopoulos, P., Mavrommati, I., Kameas, A.: End-User Configuration of Ambient Intelligence Environments: Feasibility from a User Perspective. In: Markopoulos, P., Eggen, B., Aarts, E., Crowley, J.L. (eds.) EUSAI 2004. LNCS, vol. 3295, pp. 243–254. Springer, Heidelberg (2004)

Mavrommati, I., Kameas, A., Markopoulos, P.: An Editing tool that manages the devices associations. Personal and Ubiquitous Computing 8(3-4), 255–263 (2004)

Newman, M., Sedivy, J., Neuwirth, C.M., Edwards, W.K., Hong, J.I., Izadi, S., Marcelo, K., Smith, T.: Designing for serendipity. In: Serious Reflection on Designing Interactive Systems (ACM SIGCHI DIS2002), London, England, pp. 147–156. ACM, New York (2002)

Norman, D.A.: The Psychology of Everyday Things. Basic books, New York (1988)

Schollmeier, R.: A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. In: Proceedings of the IEEE 2001 International Conference on Peer-to-Peer Computing (P2P'01), Linköping, Sweden (2001), IEEE, Los Alamitos (2001)

Streitz, N., Röcker, C., Prante, T., van Alphen, D., Stenzel, R., Magerkurth, C.: Designing Smart Artefacts for Smart Environments. IEEE Computer March 2005, 41–49 (2005)

Streitz, N., Prante, T., Röcker, C., van Alphen, D., Stenzel, R., Magerkurth, C., Lahlou, S., Nosulenko, V., Jegou, F., Sonder, F., Plewe, D.: Smart Artefacts as Affordances for Awareness in Distributed Teams. In: Streitz, N., Kameas, A., Mavrommati, I. (eds.) The Disappearing Computer. LNCS, vol. 4500, Springer, Heidelberg (2007)

Tanenbaum, A.S. et al.: The Amoeba Distributed Operating System-A Status Report. Computer Communications 14(6), 324–335 (1991)

Truong, K.N., Huang, E.M., Abowd, G.D., CAMP,: A Magnetic Poetry Interface for End-User Programming of Capture Applications for the Home. In: Davies, N., Mynatt, E.D., Siio, I. (eds.) UbiComp 2004. LNCS, vol. 3205, pp. 143–160. Springer, Heidelberg (2004)

Weis, T., Handte, M., Knoll, M., Becker, C.: Customizable Pervasive Applications. In: International Conference on Pervasive Computing and Communications (PERCOM) 2006, Pisa, Italy (2006)

Weiser, M.: The computer for the 21st century. Scientific American 265(3), 94–104 (1991)

Weiser, M.: Some computer science issues in ubiquitous computing. Communications of the ACM 36(7), 75–84 (1993)