Ambient Intelligence
G. Riva, , F. Vatalaro, F. Davide, M. Alcañiz (Eds.)
IOS Press, 2005, http://www.ambientintelligence.org

7 Computing in Tangible: Using Artifacts as Components of Ambient Intelligence Environments

Achilles KAMEAS, Irene MAVROMMATI,
Panos MARKOPOULOS

Abstract. In this essay we will present a coherent way to create UbiComp applications. These are considered to consist of tangible objects, which carry the computing and networking technology required. By providing uniform abstractions and a supporting middleware, we treat objects as components of a UbiComp application. The component architecture is made directly visible and accessible via an Editor device that enables end-users to act as programmers. The possibility to reuse devices for several purposes - not all accounted for during their design - opens possibilities for emergent uses of ubiquitous devices, whereby the emergence results from actual use and people's creativity.

Contents

7.1 Introduction

The vision of Ambient Intelligence (AmI) implies a seamless environment of computing, advanced networking technology and specific interfaces [2] [4]. Technology becomes embedded in everyday objects such as furniture, clothes, vehicles, roads and smart materials, and people are provided with the tools and the processes that are necessary in order to achieve relaxing interactions with this environment. The AmI environment can be considered to host several Ubiquitous Computing (UbiComp) applications, which make use of the infrastructure provided by the environment and the services provided by the objects therein.

Every new technology is manifested with objects that realize it. These objects may be new or improved versions of existing objects, which by using the new technology, allow people to carry out new tasks or old tasks in new and better ways. Up to now, the ways that an object could be used and the tasks it could be used for have always been determining and depending on its shape.

An important characteristic of AmI environments is the merging of physical and digital space (i.e. tangible objects and physical environments are acquiring a digital representation). As the computer disappears in the environments surrounding our activities, the objects therein become augmented with Information and Communication Technology (ICT) components (i.e. sensors, actuators, processor, memory, wireless communication modules) and can receive, store, process and transmit information; in the following, we shall use the term “artifacts” for this type of augmented objects.

Traditional objects have physical characteristics; mechanical ones also have capabilities, which describe the tasks they can do. The concept of “affordance” was introduced by psychologist J. J. Gibson [9] in order to describe the relationship that develops between objects and the tasks that can be performed with them. According to Harold Thimbleby [26] ‘… we say that an object may afford some or several sorts of action, and when it does so this is in some sense a set of natural or “easy” relations. The classic example is the door plate and door handle, which when used appropriately afford pushing and pulling the door’. Don Norman has made affordances known into the design community [22]; for him 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’. Later, Bill Gaver [8] has widened the scope of affordance to the design of interactive systems, defining affordances as ‘properties of the world defined with respect to people’s interactions with it” thus allowing for hidden affordances. Functions, which may not always be directly perceptible, can be an example of hidden affordances.

Before we explore how this concept applies to artifacts, let’s agree first that AmI artifacts differ from traditional objects in a number of properties and abilities:

- Information processing: The information that an artifact processes can be descriptions of the context of use, data to be used for a task, guidelines on how to perform a new task (i.e. a program), messages to be sent or that have been received from other objects. The result of information processing is a set of services, that is, a set of abilities that appear in the digital space and relate to information; an artifact may offer or request services
- Interaction with environment: artifacts can perceive properties of their context of use (via their embedded sensors, or by communicating with other artifacts) and can also produce responses to these stimuli (via their actuators)

- Autonomy: the operation of artifacts depends on electrical power; thus their autonomy depends on the availability of electrical power (which most of the times depends on the capacity of their battery)
- Collaboration: artifacts can exchange messages via (usually wireless) communication channels; the content of these messages may range from plain data to complex structures, including programs, database parts etc.

Artifacts possess two new affordances with respect to objects:

- Composeability: artifacts can be used as building blocks of larger and more complex systems. This is a consequence of them possessing a communication unit and requires universal descriptions of tasks and services
- 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, as they are based on the manipulation of information.

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 artifact that acts as a service consumer may seek a service producer based on a service and not artifact description. For example, consider the analogy of someone wanting to drive a nail and asking not for the hammer, but for any bearer of the hammering service (could be a large flat stone).

In this essay we will present a coherent way to create UbiComp applications. These are considered to consist of tangible objects, which carry the computing and networking technology required. By providing uniform abstractions and a supporting middleware, we treat objects as components of a UbiComp application. In the next section, we shall describe how we create digital services from the physical characteristics and properties of an object, thus providing the artifact with the two new affordances described above. In section 7.3 we discuss issues regarding people interaction with UbiComp artifacts and applications, with the help of a few indicative examples.

Then, we introduce Gadgetware Architectural Style (GAS), a set of concepts, middleware and tools that we developed, which enables people to compose UbiComp applications by combining the services offered by artifacts. In section 7.5 we describe a concrete example of using GAS to create a UbiComp application by taking advantage of the two new affordances of artifacts. These new affordances are indicated to people via a third device, the Editor, which is described in the next section.

In section 7.7 we discuss the applicability of GAS concepts and the ways that this affordance of component-connectivity can also be indicated by elements of the physical design of object, such as physical, auditory, haptic or visual design elements for control and feedback that visualizes the interface of the digital self of the object. Then we present briefly the outcome and conclusions we drew from the evaluation sessions we held with experts and end-users, regarding the applicability and usability of GAS. In the last section we summarize the benefits of our approach in comparison with other similar approaches.

7.2 Building Artifacts

Turning an object into an artifact is a process that aims at enhancing its characteristics and properties and abilities so that the new affordances will emerge. In practical terms, it is about providing the object with the necessary hardware and software modules. Broadly it consists of two phases:

1. Embedding the hardware modules into the object
2. Installing the software modules that will determine its functionality.

This stepwise process is described here briefly, as we tried to avoid technical overloading. It can be universally applied to every object and at the same time takes into account the object's physical properties. In the end, it provides the object with a "digital self", that is, a representation of it into the digital space that can be perceived by other artifacts.

7.2.1 Phase one: embedding the hardware modules into the object

Generally, this phase involves embedding into the artifact a power source, an array of sensors and actuators, a processor board, a wireless module, a few buttons and a screen.

Nevertheless, some of these modules may not appear in all artifacts. For example:

- Artifacts, especially mobile ones, may have batteries but large artifacts can be directly connected to an electricity socket
- The number of sensors may range from tens to … none! An artifact may simply receive sensor data from other artifacts
- The processor board may be embedded in the artifact, but it may also be the case that the artifact "rents" processing time and storage space at a nearby (using network, not physical terms) server
- The wireless module is probably the most distinguishing module; nevertheless, an artifact may only carry a passive tag, which, when tracked, triggers an action at another artifact
- Screens and buttons are generally avoided, especially in small size artifacts, as they occupy a lot of space and consume a lot of power.

Most usually, the hardware modules will be embedded in the artifact (Figure 7.1) and unique ID (could be a serial number) will be assigned to it at the time of its manufacture.

This will improve the "oneness" of the artifact. However, as AmI technology matures, an increasing number of artifacts will be "virtual", in the sense that they will be composed from distributed objects on a service description basis. As every object (and thus, hardware module) acquires a digital representation based on its capabilities and properties and the services it offers or requests, then an artifact may embed a specific number of hardware modules and attempt to locate other modules based on a service-based description.

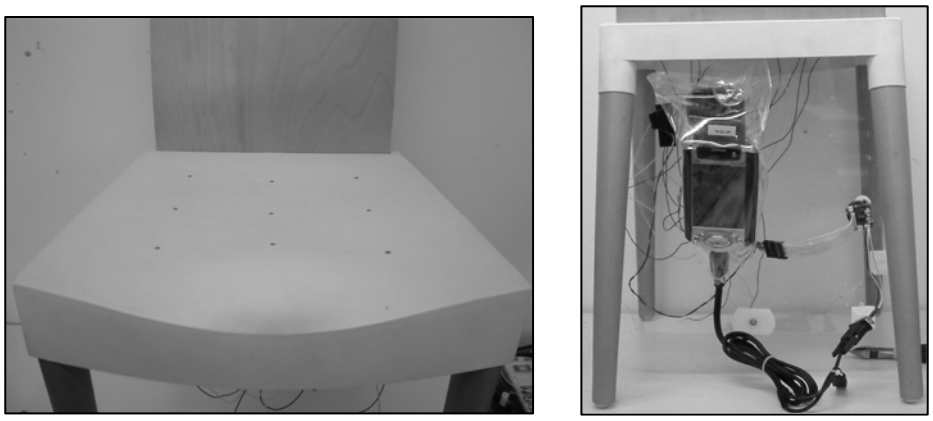

Figure 7.1 Embedded sensors in the surface of an augmented Chair and the hardware modules underneath it: a 25mm FPGA Signal conditioning/ RS232 and coin cell modules can be seen

In the terms of the present discussion, the mobile phone can be considered as one of the first AmI artifacts to appear. As an artifact the phone is equipped with a processor board, a wireless communication unit, a screen and a set of buttons (we are not referring to the numeric pad, but to the buttons used to operate the services of the phone). Future phones may also be able to use a nearby screen, e.g. a television or public display; or audio output capabilities, e.g. a home audio system or an office intracom. This will be possible when a universal (standardized) description of their services becomes available. If we generalize this concept, we can imagine a situation where a range of services e.g. in the case of the phone dialing, speaking, listening, reading, etc., are made accessible through a dynamic, personalized or optimized for context set of artifacts, e.g., the remote control, the mobile phone or even decorative artifacts providing a tangible interface to the telecommunication service.

7.2.2 Phase two: installing the software modules

The hardware modules (i.e. the resources) of an artifact are being used by its software. The following software modules need to be installed in an artifact:

- Hardware drivers: used to manage information exchange between the operating system and the external resources (i.e. sensors, actuators, screen etc)
- Networking subsystem: it is responsible for the exchange of messages between artifacts
- Operating system: manages the use of resources and translates service requests into commands to resources and vice versa
- Middleware: provides abstractions that enable an artifact to be part of a distributed system.

This layered architecture resembles a computer, only in this case emphasis is placed on the role of middleware. In fact, many of the architectural designs and technical solutions that have been applied to traditional computers are being ported to the requirements of artifacts. The success of the attempts depends on the ability to adapt to the resource constrained and modular nature of the artifacts.

Considering the networking subsystem as independent enables designers to decouple the messaging protocol (which is part of the operating system or middleware) from the

networking protocol (in fact, an artifact may use more than one networking protocols, i.e. Bluetooth, InfraRed etc).

7.2.3 Some issues to discuss

When turning an object into an artifact, there are two major issues one has to deal with: robustness and heterogeneity. There are several other issues as well, but we believe that these two are the most important, because the first one relates to the operation of the artifact as a unit and the second one to its ability to become part of ubiquitous applications.

AmI artifacts are complex systems, which include advanced electronic modules. The requirement that each module operates reliably is not good enough, as the entire system must be reliable as well. Currently, reliability largely depends on the natural wear of the hardware modules; in the future, when AmI artifacts will become a set of service descriptions, it will mostly depend on the availability of infrastructure and its ability to deliver the services.

Another aspect of the same problem is serviceability of artifacts, which with respect to hardware translates into an easy way to replace hardware parts when they fail. Such parts may be the battery, sensors / actuators, or more complex modules, such as the screen.

Regarding design, this calls for increased modularity and universal module interfaces, as well as for a novel design of the object part of the artifact, which will provide easy access to some of the hardware parts.

Finally, if artifacts are to be autonomous and exhibit coherent functionality, then a set of hardware controllers are needed. These will process sensor data into information required by the software and vice-versa. The design of these controllers has to be generic enough (so that one controller can be applicable to a number of different similar artifacts) yet efficient. Currently, solutions based on Field Programmable Gate Arrays (FPGAs) are being investigated.

On the other hand, heterogeneity is primarily a result of the component-oriented approach adapted at almost all levels of an AmI environment: an artifact is composed of modules; an application is composed of services and artifacts, etc. At the artifact's level, which is our concern here, the manufacturers have to ensure that all hardware components co-operate smoothly with the software. To this end, hardware drivers will be included in the artifact at the time of manufacture, together with the respective hardware resources.

The use of hardware drivers as an intermediate layer between the hardware and the operating system enhances modularity, as new drivers may be downloaded or new hardware resources may be added, without affecting the ability of the artifact to function. The key requirements are efficiency and compatibility with the operating system.

7.3 Interaction and use

Within an AmI environment, people will use several UbiComp applications and will interact with different artifacts and services. In addition, a UbiComp application is usually perceived as a collection of interacting artifacts. From the interaction point of view, we are mostly concerned with the interface of the artifacts and the applications; most of the times this shall directly affect or depend upon the physical form and shape of the artifacts. In addition, as an artifact participates in a configuration by being physically present, it promotes heterogeneity in design, which is a desirable characteristic of UbiComp

applications. Thus, one claim that people are now enabled to "engineer" new purpose-specific systems by using artifacts in their environment.

7.3.1 Levels of Interaction

Interaction in an AmI environment has to be considered from various aspects. With respect to "intention", people interact with an AmI environment in order to:

- Engineer a UbiComp application within the environment: this application usually is a composition of artifacts, which collectively serve a specific purpose or satisfy a declared set of needs. To this end, people have to be equipped with tools in order to create, edit, destroy, and debug/fine-tune the application. For example someone may set an automation to have the window blinds open up when the alarm clock rings, so that (s)he may wake up with daylight in the room (Figure 7.2)
- Use an application to satisfy their needs: such an application may be composed by people themselves, or could be bought and installed. In the latter, issues regarding the way people can cope with unanticipated operation may arise. In the former, as people interact with the application (by using the artifacts that compose it), the need for the application to "learn and adapt" may arise. One could buy one video-projector for example, that has the preset ability to close the window blinds and dim the lights when it is on, so that viewing is done under appropriate lighting conditions. In the case that the person who buys it does not wish for all the lights to dim, he can edit the supplied functionality with an editing device (Figure 7.3), and choose for example the floor and desk light not to dim out, while letting the main room lights to switch off completely. Over time he may also choose to change the function given by making a different configuration whereby the sound of all other devices (stereo, PC, telephone, etc) is set to mute when a film is being projected.

With respect to "interacting parties", interaction takes place in two levels:

- Artifact-to-artifact: the objects themselves may form an "underlying" layer of interactions, mainly in order to exchange data and to serve their purpose better. Such interactions, in the previous example, occur between the alarm clock and the window blinds. The degree of visibility and control that people may have on these interactions can vary depending on people's ability to perceive the system state

Figure 7.2 Scenario of artifacts associated with each other, in the ubiquitous home: when the alarm clock rings, fresh coffee starts being made

- User-to-environment. The user interacts a) With any single artifact b) With a collection of co-operating devices. In the previous example while the user physically interacts with the alarm clock, he is also interacting with the window blinds associated to the alarm clock. The intention behind each interaction may vary; moreover, if one artifact participates in many different applications, then a policy has to be devised to solve possible conflicts.

Figure 7.3 A user can create certain associations between artifacts

Any of these two types of interaction may happen either explicitly or implicitly. An explicit interaction happens under the control of people always provides feedback about its state to them. Although this may seem desirable, it may also become very annoying if one takes into account that there will be hundreds of artifacts in our environment. Implicit interactions are usually under the control of actors other than people; these could be processes, artifacts, intelligent agent mechanisms or even artifact owners. Implicit interactions can only be acceptable if they can be trusted and do not violate privacy or ethics.

Finally, one should also consider the context of interaction, which ranges from public to private (with respect to disclosure), from individual to shared (with respect to stakeholders) and from closed to open (with respect to space).

7.3.2 Issues of Use

Each object that participates in our everyday world has been designed with certain tasks in mind. The ways that we can use an ordinary object (sometimes implied by the "object's affordances") 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: not only have the objects been designed to be suitable for certain tasks, but also their physical properties constrain the tasks people use them for. As everyday objects are "enhanced" with computing and communication capability, the user has to learn the new ways that they can be used (indicated by designing new affordances) and the tasks they can participate in.

Thus, the ubiquitous computing paradigm introduces several challenges for human-computer interaction. People can then act upon their environments, be it physical or enhanced Ubiquitous Computing environments, by setting goals, forming plans and perceiving results. At the cognitive level, the disappearance of the computer forces people to form new mental models about their tasks that involve objects and environments (that now may start to involve using hidden IT capabilities). On the other hand, if the appearance and function of everyday objects / environments change (or new objects appear into our everyday life), then people will have to adapt or form new models of tasks involving these objects.

As the Human Computer Interface penetrates the world of physical, tangible objects, the direct manipulation paradigm will have to include metaphors describing interaction with tangible objects and collections of objects, as well as with computational tools. Thus, objects may re-appear in the task model of people, as they try to get accustomed to their new capabilities and possible uses. Then, in order to interact with them, different options have to be investigated:

- Usage of screen-based interfaces: screens can be embedded in objects or may be located in their vicinity. The properties, state and functions of artifacts can be projected onto a screen. A command-based interface can be used by people to interact with the artifact. Clearly, for most everyday objects this would constitute a significant alteration of their form; if such a paradigm is chosen, then it is better to use objects that already embed a screen, such as a TV or a PDA. Issues regarding privacy and space have also to be considered
- Embed interaction widgets in the artifact itself: in this case, the object's external form is enhanced with buttons or LEDs, which enable certain new functions to be carried out and provide feedback about the object's state. Such an approach will alter the appearance of the object and thus could cause mistrust to people. However, the case of cellular phones shows that a gradual introduction of such widgets may increase the chance of acceptance
- Embed computation and communication abilities in the material of the artifact: in this case, the artifact is a computer, as computation becomes part of its nature. Such an approach relies on miniaturization of the computational modules (in an approach similar to Smart Dust [27]), but is not yet technically feasible, while it depends heavily on the availability of a network infrastructure.

Motivating factors for people to use this technology may include new, enhanced services and tasks made possible; better response rate in ordinary services; savings in effort and time in carrying complex tasks; low intrusion of the new gadgets in existing task models; steep learning curve of the new affordances; and trendy design and appealing shape.

Living with and using UbiComp artifacts may not seem easy at first, and may require certain new skills to be developed (including abstractions and models to reason about them). Nevertheless, it may be the case, as it is for example with writing, or with riding a bicycle, that once the skill -however complex it may be- is acquired, over time it feels natural, easy and transparent in use. The design of the object's form and physical properties will also affect the interaction. In fact the design of objects, which constitutes their interface, may have to be reconsidered so that their new capabilities can be promoted to the user (indicated by appropriate elements for the nature of each object).

As we are only starting using ubiquitous technologies, we apply them with applications of the past in mind. To quote McLuhan [20] "we look at the present through a rearview mirror. We march backwards into the future".

We can assume that, as it is often the case with fundamental technology advances [23] once we get more accustomed to the new medium, people would come up with applications and uses that could not be previously perceived. Later generations of UbiComp applications may be nothing close to what we can currently imagine.

7.4 GAS: The computing in-tangible style

The Gadgetware Architectural Style (GAS) constitutes a generic framework, shared by users and designers, for consistently describing, using, reasoning about UbiComp applications within the AmI environment (Figure 7.4). GAS provides:

- A vocabulary that can be used to describe the artifact collection; the vocabulary should appear "natural" to the users and would build upon notions already used in a similar context
- A framework for the interpretation of the collection (that is, for assigning meaning to the vocabulary). Such a framework could be implemented as an ontology shared among all artifacts. This ontology could be freely available, while ontologies describing specific applications or systems could be copyrighted
- Rules that will define the correctness of the artifact collection or constrain it. These would in fact form a programming language, which would appear to people as natural as using the objects in the environment. A correct representation of context would be required to make such a system usable.

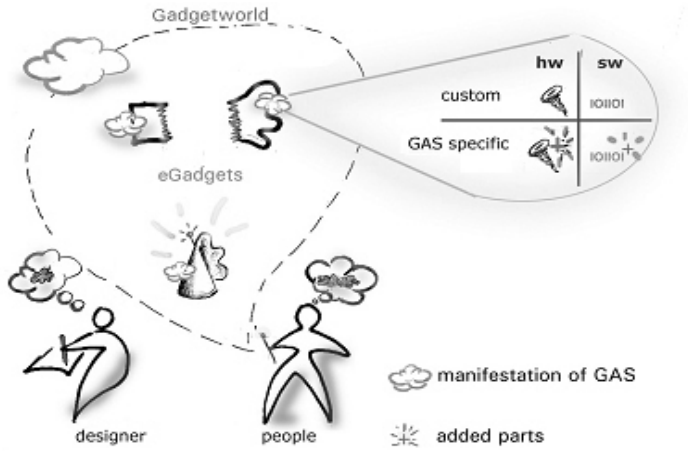

Figure 7.4 GAS, an Architectural style that acts as a common referent between people, designers, and artifacts

GAS has been developed in project e-Gadgets [3] and consists of a set of concepts encoded in an ontology, the Plug/Synapse model, the middleware that implements it, a methodology for developing artifacts and a set of tools that enable people create and manage UbiComp applications (Figure 7.5).

The basic concepts underlying GAS are [15]:

- *eGadget:* Generally speaking, eGadgets are everyday tangible (physical) objects enhanced with sensing, acting, processing and communication abilities. Moreover, processing may entail "intelligent" behaviour, which can be manifested at various levels. In the GAS context, eGadgets can be regarded as GAS-aware artifacts, which are used as building blocks to form eGadgetworlds

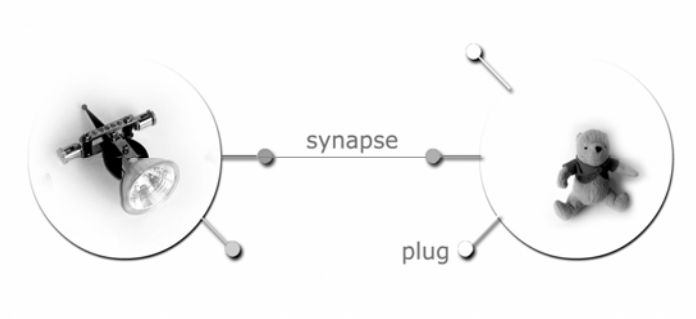

Figure 7.5 The Plug Synapse model: The artifacts' capabilities (Plugs) can be inter-associated with invisible links (Synapses) to form ubiquitous computing applications

- *Plugs:* They are software classes that make visible the capabilities, properties and services of eGadgets to people and to other eGadgets. They may also have tangible manifestation on the eGadget's physical interface, so that users can utilize them in forming Synapses
- *Synapses:* They are associations between two compatible Plugs
- *Gadgetworld:* A Gadgetworld is a specific configuration of associated eGadgets, formed purposefully by an actor (user or other), which communicate and / or collaborate in order to realize a collective function. Gadgetworlds are our notion of UbiComp applications: dynamic functional configurations of inter-related eGadgets exhibiting collective behaviors.

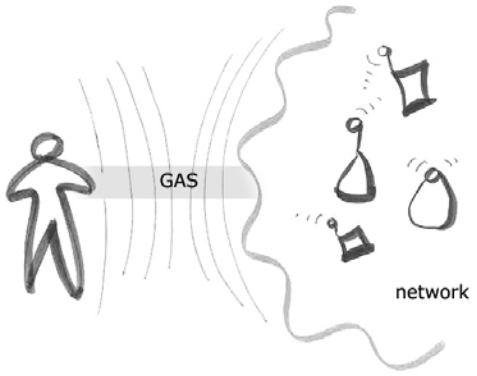

Figure 7.6 GAS middleware shown here as a layer between user and system

GAS-OS is the middleware that runs on every eGadget and implements GAS concepts (Figure 7.6). GAS-OS is a component framework that manages resources shared by e-Gadgets, it determines their software interfaces and it provides the underlying mechanisms that enable interaction among e-Gadgets. GAS-OS supports the composition of e-Gadgets, without having to access any code that implements their interfaces.

GAS-OS offers following services to the user and to other eGadgets:

- *Plugs discovery and advertising:* The GAS-OS of a specific eGadget is responsible for the discovery of all other Plugs (and consequently eGadgets) within range. GAS-OS multicasts a *hello message* and all listening eGadgets, i.e. all eGadgets within range, respond to it by sending an XML-based advertisement. This advertisement contains all the data that one eGadget can know about the other [14]

- *Synapse establishment – disestablishment:* GAS-OS enables the user to form Synapses between Plugs and also to destroy them. It must ensure that Plugs get connected only when they are available for connection and only if the two connecting Plugs are "compatible". The ontology is used to define the degree of compatibility. Then, it takes care of the handshaking between the two connecting Plugs until the Synapse is established. GAS-OS provides the means for successful connection-disconnection ensuring that these procedures are executed as atomic ones that either success or fail before releasing the Plugs. Moreover, it ensures that the Plugs will not stay locked for infinite amount of time, in the case where a Synapse establishment fails. The Plugs are fully functional and do not stay locked during this procedure. This is very important because network delay can be long even for successful Synapse establishments. Also, it is the GAS-OS responsibility to ensure that when an eGadget is shutting down all Plugs connected to its own Plugs are notified and all Synapses are dropped. Finally, the GAS-OS ensures that an eGadget on startup will attempt to reestablish the Synapses of the Gadgetworlds it participated in when it was shutdown
- *Synapse management:* eGadgets are notified about changes in the state of other connected eGadgets via the Plugs. Thus the GAS-OS is responsible for sending and handling these notifications and forwarding them to the computational logic of the eGadget. Thus the GAS-OS acts as a mediator in the eGadgets collaboration.

A Plug is an abstraction of the properties and abilities of an eGadget. It is implemented as a record and contains attributes and methods, which implement the ways it can be used (protocol), the service it can offer (methods) and its state (attributes). In fact, it is the only way other eGadgets can use eGadget services and have access to the eGadget properties. A Plug is accessible to other eGadget modules via GAS-OS only, thus providing a unified way to access the resources of a Gadgetworld. Plugs have a direct relation to the sensors/ actuators and the functions implemented in the eGadget by its manufacturer.

Procedurally, a Gadgetworld is formed as a set of Synapses. Once a Synapse is established, the involved eGadgets interact on their own, independently and transparently of the existence of Plugs. A Gadgetworld should be considered as being always operational until explicitly disassembled by the user. When switched on, each eGadget constantly attempts to re-establish its Synapses. Each end of a synapse is managed by the GAS-OS running on each e-Gadget, thus implementing a peer-to-peer architecture. A synapse serves as the abstraction of a communication channel between peers. However, this occurs only when they have 'discovered' each other. Discovery is twofold: (1) on demand by the editor and (2) proactively carried out by an e-Gadget, after a synapse request.

On a Gadgetworld level, the heterogeneity of the devices is a major issue, because eGadgets of the same type can be created by various manufacturers. As eGadgets are autonomous, they have to inform other eGadgets about their abilities, properties and services, in a commonly understood, consistent and unambiguous way. In order to enable interoperability, the eGadgets have to use the same language and a common vocabulary (although each may implement a different mechanism to interpret them). The GAS Ontology [7] can provide this necessary common language for the communication and collaboration among eGadgets as it describes the semantics of the basic terms and defines their inter-relations. In addition, the ontology provides specific rules for plugs compatibility and eGadgets replacement feasibility, by providing descriptions of various services provided by the eGadgets. This solution allows each eGadget to have a different

ontology with the condition that all ontologies are based on a common vocabulary. Thus, although all eGadgets have different knowledge, it is represented with terms and concepts common to all the eGadgets. According to this solution the GAS Ontology is divided into the following two layers: the GAS Core Ontology (GAS-CO) and the GAS Higher Ontology (GAS-HO). The GAS-CO provides eGadgets with the necessary common language that they need in order to describe their acquired knowledge represented by the GAS-HO.

7.5 Use of artifacts into functional combinations: an example

Let's take a look at the life of Patricia, 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, the "extrovert Gadgets". 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 e-Desk (it could sense light intensity, temperature, weight on it), an e-Chair (it could tell whether someone was sitting on it), a couple of e-Lamps (one could remotely turn them on and off), some e-Book tags (they could be attached to a book, tell whether a book is open or closed and determine the amount of light that falls on the book), and …. an e-Carpet (you just had to step on it). Pat had asked the store employee to pre-configure some of the e-Gadgets, so that she could create a smart studying corner in her living room. Her idea was simple (she felt a little silly when she spoke to the employee about it): 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 (she hadn't thought of any use of the carpet, but she liked the colors).

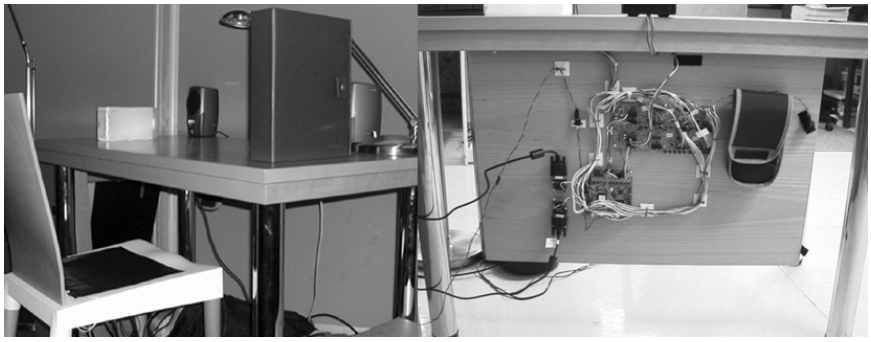

Figure 7.7 The surface of e-Desk and the hardware modules underneath it

The behavior requested by Pat requires the combined operation of the following set of artifacts: e-Desk, e-Chair, e-Lamp and e-Book. All these eGadgets can be created by attaching hardware and software modules to ordinary objects. For example, in order for the e-Desk to be able to sense weight, luminosity temperature and proximity, it has to be equipped with pressure pads, luminosity sensors and an infrared sensor (Figure 7.7). Pressure pads are mounted underneath its top surface and cover all of it. Luminosity and temperature sensors are evenly distributed on the surface. Four infrared sensors are placed on the legs of the table and another one on its top. All sensors and actuators interface with an FPGA that collects and filters sensor readings into the GAS-OS middleware, which

currently runs on an iPaq attached to the object (to preserve autonomy of objects, we have used one iPaq per artifact, which provided the required processing and communication hardware).

Once the hardware is installed, one has to install GAS-OS in the artifact's iPaq and configure it to interface with the hardware. This is achieved via a special software module, the Gadget-OS, which interfaces with an FPGA that drives the sensors. When these objects have reached beyond prototype stage, all necessary hardware (including sensors, processor, wireless module, battery, boards and circuitry) will be embedded into them during their manufacture.

The collective function of this application can be described as:
When this CHAIR is NEAR the DESK
AND
ANY BOOK is ON the DESK,
AND
SOMEONE is sitting on the CHAIR
AND
The BOOK is OPEN
THEN
TURN the LAMP ON.

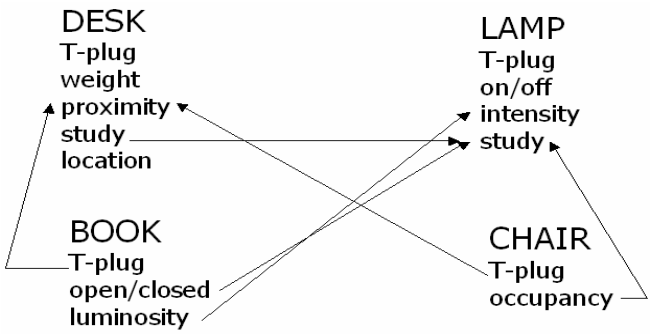

Figure 7.8 Schematic representations of the connections between appliances, in the above scenario

In order to achieve the collective functionality required by Pat, the employee in the store had to create a set of Synapses among e-Gadgets' Plugs (Figure 7.8). This type of functionality and component structure is created, inspected and modified through the Editor. For example, Pat can subsequently define the intensity of the e-Lamp when it's being automatically switched on; thus the light won't blind her. Or, if an intelligent agent is used, it could adjust each time the light intensity based on the overall amount of light in the room, as it is recorded by luminosity sensors distributed on objects in the room.

7.6 Tools for end-user programming

People can supervise the functional capabilities of devices in one's environment via other specialized devices, the Editors [18]. An Editor is a facilitator device that can be used for creation and editing of UbiComp applications. Since an application can be regarded as a collection of associations (Synapses), the main role of the Editor is to allow the user to

create and edit Synapses. Since a Synapse is established between two Plugs, all the user has to do is to indicate which two Plugs to associate; the remaining actions can be performed by the Editor via the services offered by GAS-OS.

Editors can provide an integrated user interface, be it graphical or other; potentially, they may be linked to multimodal interfaces integrated in the environment, such as speech or gesture input systems. With Editors it is possible to supervise and interface with the various artifacts (eGadgets) in ones environment. One can supervise the inter-connectable capabilities of an appliance (the Plugs). People can act-upon these capabilities by associating them together into matching pairs that serve specific functions (the Synapses), they can break existing associations, pause them, or add parameters in the association to influence the details of the function. The associations created/edited are not stored locally, as would be the case with a centralized system, but are stored in the artifacts of the AmI environment. Thus, in the case of failure (such as object movement beyond network range) much of the application functionality can be restored.

The editing capabilities can be used by *designers* to create UbiComp applications, without having to start from scratch, as they may (re)use existing artifacts as components of applications. Editing may be used by *end users* to personalize UbiComp applications they get ready-made, or for practicing their own creativity in creating unforeseen associations between artifacts for achieving niche functions. The Editor's core functionality can also be accessed directly by *intelligent agents* that construct and adapt applications by monitoring user behavior.

7.6.1 Editing functionality

The Editor is used in order to indicate/make visible the available eGadgets and Gadgetworlds, form new Gadgetworlds, assist with debugging, editing, servicing, etc. It identifies the style-compatible artifacts, as well as the current applications that are available and displays them to people. Plugs have a direct relation to the sensors / actuators and the functions intended by the artifact's manufacturer. Synapses, the links between the compatible Plugs of eGadgets, are visualized and manipulated with the Editor.

Plug identification and selection is a task that depends heavily on the user expertise. A novice user might not be interested in or understand more than just the description of the Plug and hence base his/her selection on the natural language description provided by the Plug implementer. Experienced users and computer experts on the other hand, may prefer to have all the technical details to facilitate a proper selection of the Plug. Creation of the required number of Synapses eventually results in the formation of a new application (Gadgetworld). Once a Synapse is established the part of GAS-OS that runs on each artifact (eGadget) will ensure proper working. The application (Gadgetworld) remains operational as long as required by the user (unless there's technical inability to maintain its functionality) and finally it might be deactivated or broken down by destroying its Synapses. Based on these requirements, the Editor should provide the following basic services to the user:

- Discovery of eGadgets and Plugs
- Information about the discovered eGadgets, Plugs, Synapses (and their properties), and Gadgetworld applications
- Supporting the user in creation, editing and destroying a Synapse.
- Creation of Gadgetworlds (as a collection of Synapses)

- Gadgetworld operation (activation, deactivation, elimination) and management (i.e. editing).

In addition to the above basic functions, the Editor can provide classification of eGadgets, Plugs, Synapses, and Gadgetworlds as well as services that facilitate the selection and manipulation of these entities. It can also facilitate the user in supervising and debugging Gadgetworlds. An intelligent editor for example may be able to provide suggestions based on other similar configurations made, or give step by step context related guidance and tutorials.

7.6.2 Editor Interfaces

The Editor provides its complex functionality through the user interface (Figure 7.9), which, in turn depends on the capabilities of the device that the Editor software is running on. A software module for example running on a PC or Laptop can offer the editing functionality via graphical user interfaces. Many other possible interfaces can be considered for the Editor. Several interaction and interface design issues and challenges arise, that have to be resolved; many of those influence the design of GAS-OS.

Figure 7.9 Graphical user interface of the editor on a PDA

The Editor is a software that can run on top of an existing information appliance (such as a PDA, or a PC). While the top software layer utilizes the particular interface resources of the appliance it is residing at, its core is independent from the device it runs on allowing for a multitude of interaction modalities to be implemented.

One Editor has been developed for the desktop PC environment which allows the standard windows Graphical User Interface features (such as drop down menus, drag & drop, etc). Two different interface implementations have been created using on-screen interfaces, on a handheld computer and one on a personal computer (laptop). This enabled testing the Editor and conduct end user evaluations of the system and its concepts.

7.7 Applicability of the Gadgetware Architectural Style approach

The applications that can be created with this model may vary. Objects can range from small scale (keys, lights, door handles) to bigger ones (Stereos, TV sets. desks, carpets) up to large ones (such as rooms, buildings, city squares, etc). By associating these home devices together into collaborating collections, people can shape their own environment. Such an approach supports the development of open systems [19] and at the same time can

be used to explain the system to people; this is necessary in order for them to be able to manipulate such environments. Tailor made small scale applications (devised by people for their niche needs), home automation, and control, larger scale applications for buildings (i.e. safety and security in communal buildings or hotels), physical games and collaborating toys can all be addressed by this model.

One assumption underlying GAS is that appliances and objects in the home are manufactured to be autonomous and functionally self-contained. Moreover, they can locally manage their resources (processor, memory, sensors/actuators etc). GAS provides a compatibility framework that is a layered middleware that enables them to share definitions of services, exchange data, interpret incoming messages correctly and act accordingly. Thus GAS enables artifacts to be used as components of a non-a priori defined system; at the same time enables users to play an active role in understanding and defining the functionality of their ubiquitous environment. For this GAS provides a middleware that can directly interface with hardware components and also serves as a layer on top of existing network protocols or distributed system architectures, enhancing them with GAS-specific functionality.

The solution we propose is communication (hence the term “extrovert“), as opposed to mere message exchange. Communication, according to Habermas, aims to achieve a shared understanding [11]. He says that one communicates in order to make known a desire or intention; then others can respond to the suggestion. In the e-Gadgets approach, a Synapse is formed as a result of negotiation among eGadgets. Negotiations and subsequent data exchange are based on ontologies possessed by the eGadgets; the only intrinsic feature of an eGadget is the ability to engage in structured interaction.

In principle, this approach can scale both “upwards” (towards the assembly of complex, distributed eGadgets out of simpler ones) and “downwards” (towards the decomposition of eGadgets into smaller parts). It treats evenly powerful eGadgets (having their own processing and communication (to be considered as tagged artifacts, which borrow processing and storage capabilities from a server).

7.8 End User Evaluation

The end-user that would engage in editing applications in one environment is considered to be a ‘technophile’ but not a programmer; experts from User System Interaction that matched this profile performed a set of evaluation activities during several evaluation sessions (Figure 7.10). The approach of considering artifacts as components to create UbiComp applications was evaluated in the course of user and expert trials [17]. An expert review workshop and an analysis based on the Cognitive Dimensions framework [10] were done to assess the concepts in preliminary phases of the prototype implementation.
Feedback was also collected using a hands-on Demonstration (with three artifacts and one PDA based editor), that was shown in 2 conferences in 2003 (DC-Tales and BSCHI [16]).

29 completed feedback questionnaires were received from both these events, while aprox.100 people visited the stands spending 5-10min. to experience themselves this technology. Finally, a short user evaluation was held in iDorm, a specially constructed student dormitory that has been set up within a computer laboratory at the University of Essex, and is equipped with several sensing and actuating components, which for this study were controlled through GAS-OS. The study was a combination of short user tests (6 users in pairs have used eGadgets and an Editor to create applications in the iDorm; they had 2 hours per pair) and a single trial that took place overnight. This evaluation aimed to

gauge how potential users grasp the concepts and whether they can create or modify their own ubiquitous computing applications.

In the first evaluations, skepticism was noted among HCI experts regarding the ability of end-users to grasp the concepts proposed. The short user-tests seem to rest the fears of an impossible-to-use complexity. Although scaled up and longer user tests are required, to gain more confidence in this conclusion, this can hold true especially for new generations of users growing up surrounded by technology. All evaluation participants that had a hands-on experience using this technology familiarized with the concepts very quickly, within only a few minutes (aprox. 5 min.) of explanation. The majority of subjects succeeded in creating simple applications for themselves (using 2-3 objects with 2-3 connections), using the Editor provided, in spite of the fact that the editor interface was at a very preliminary stage (the editor was used as a functional tool so as to test the research concepts, providing an appropriate level of robustness).

Figure 7.10 Participants in a user trial, configuring a ubiquitous application

In the first evaluation session a broader set of issues was addressed, relating to the component based approach for the UbiComp home applications. A subset of the evaluation tasks concerned the Editor. Some of the most recurring themes of the discussion were:

- Ubiquitous computing technologies embedded in physical objects add hidden behavior and complexity to them. Problems may arise if this behavior is not observable, inspect able and predictable for the user
- Intelligence causes problems of operability and of unpredictability for users. It must be used with caution and this should be reflected in the demonstrations built
- Constructing and modifying applications in the home equipped with UbiComp applications is a problem solving activity performed by end-users. As such, it has an algorithmic nature and thus good programming support should be offered.

Following the last observation, the expert conducting the evaluation has conducted a further assessment using the Cognitive Dimensions framework, a broad-brush technique for the evaluation of visual notations or interactive devices. Some of the most interesting points resulting from the evaluation with the cognitive dimensions framework were the following:

- There will always be an initial gap between their intentions and the resulting functionality of a user composed application. Users will have to bridge this gap based

on the experience they develop after a trial-and-error process. An Editor can shorten this initial gap, by allowing several different ways of expressing the user's goals

- Since an object can be part of several in-home applications at the same time, the effect it has on each is not easy to understand from the physical appearance. Developments of the Editor will need a way to illustrate to the user how the specification of the parts influences the dynamic behavior of the whole application (similar to debuggers in Object Oriented environments)
- Editors should aim to bridge the gap between architectural descriptions of an application and the user's own conceptualizations, which might be rule-based, task oriented, etc. (improving the Closeness of Mapping dimension)
- To create and edit UbiComp applications only a few conventions need to be learnt by the end user (appropriately low terseness)
- The Editor should make observable any logical dependencies between seemingly unrelated physical objects (hidden dependencies). There are side effects in constructing applications. A state change in one component may have non-visible implications on the function of another. Some way of visualizing and inspecting such connections needed to be present
- An object can belong to several applications, the effect it has on each is not easy to understand from the physical appearance (Role Expressiveness). One solution is to represent this via the Editor)
- The abstraction level is appropriate for the target user audience (Abstraction Gradient Dimension).

The evaluation feedback can be summed up as follows:

- The application behavior should not surprise the user, i.e. automation or adaptation actions should be visible and predictable (or at least justifiable)
- End-users acting as UbiComp application developers should be supported with at least as good tools as programmers have at their disposal, e.g., debuggers, object browsers, help, etc
- Multiple means to define user intentions should be supported by the graphical interface of the Editor, as people conceptualize and express their intentions in a variety of ways, (which are not necessarily structural abstractions of the system)
- The acceptability of the Gadgetworld concepts depends on the quality of the actual tools and their design.

The short user-tests seem to rest the fears of an impossibly complexity, especially for new generations of users growing up surrounded by technology. Scaled up user tests are required, both in scope and duration to gain more confidence in such a conclusion. In the conducted evaluation sessions the feasibility of letting end-users architect component based ubicomp applications was demonstrated. The experiences reported suggest that an architectural approach where users act as composers of predefined components or by interacting with intelligent agents are two worthy and complementary approaches.

7.9 Conclusions

Several research efforts are attempting to design ubiquitous computing architectures. In the context of the Disappearing Computer initiative, project "Smart-Its" [13] aims at developing small devices, which, when attached to objects, enable their association based on the concept of "context proximity". Thus, the collective functionality of such a system is mainly composed of the computational abilities of the Smart-Its, without taking into account the "nature" of the participating objects. A more complete and generic approach is undertaken by project "Oxygen", which enables human-centered computing by providing special computational devices, handheld devices, dynamic networks and other supporting technologies [5]. Another interesting project is "Accord", which is focused in developing a Tangible Toolbox (based on the metaphor of a tangible puzzle) that will enable people to easily embed functionality into existing artefacts around the home and enable these devices to be integrated with each other [1]. Other related research efforts are:

- Gaia [24] provides an infrastructure to spontaneously connect devices offering or using registered services. Gaia-OS requires a specific system software infrastructure using CORBA objects, while mobile devices cannot operate autonomously without the infrastructure
- BASE [6] is a component-oriented micro-kernel based middleware, which, although provides support for heterogeneity and a uniform abstraction of services, the application programming interface requires specific programming capabilities by users
- TinyOS [12], an event driven operating system, designed to provide support for deeply embedded systems (i.e. sensor networks), which require concurrency intensive operations while constrained by minimal hardware resources.

The overall innovation of the Gadgetware Architectural Style (GAS) approach lies in viewing the process where people configure and use complex collections of interacting eGadgets, as having much in common with the process where system builders design software systems out of components. This approach regards the *everyday* environment as being populated with tens even hundreds of artifacts, which people (who are always in control) associate in ad-hoc and dynamic ways.

Then, GAS-OS, the software that implements GAS, can be considered as a component framework, a collection of software components and architectural styles that determines the interfaces that components may have and the rules governing their composition [25].

GAS-OS manages resources shared by eGadgets, and provides the underlying mechanisms that enable communication (interaction) among eGadgets. This approach builds on the premise that GAS could offer to the ubicomp systems domain the benefits demonstrated in the domain of software engineering. For example, the proposed concept supports the encapsulation of the internal structure of an eGadget and provides the means for composition of an application, without having to access any code that implements the interface. Thus, this approach provides a clear separation between computational and compositional aspects of an application, leaving the second task to ordinary people, while the first can be undertaken by experienced designers or engineers.

The benefit of this approach is that, to a large extent, system design is already done, because the domain and system concepts are specified in the generic architecture [21]; all people have to do is realize specific instances of the system. Composition achieves adaptability and evolution: a component-based application can be reconfigured with low cost to meet new requirements. The component architecture is made directly visible and

accessible via the Editor. This enables end-users to act as programmers. This end user programming approach may be especially suitable for Ubiquitous computing applications.

The possibility to reuse devices for several purposes - not all accounted for during their design - opens possibilities for emergent uses of ubiquitous devices, whereby the emergence results from actual use.

The proposed model is easily comprehendible; therefore by the appropriate use of tools, the e-Gadgets technology can be usable by designers of UbiComp applications, but also by untrained end-users. Subsequently this approach opens possibilities for emergent uses of artifacts whereby the emergence occurs from people's own use. Potentially it can enable the acceptability of UbiComp technology into people's environments, as well as enable the making of emerging niche applications.

References

[1] Accord project website: http://www.sics.se/accord/home.html.

[2] Disappearing Computer initiative. Online: http://www.disappearing-computer.net/.

[3] e-Gadgets project website: http://www.extrovert-gadgets.net.

[4] ISTAG website: http://www.cordis.lu/ist/istag.htm.

[5] Oxygen project website: http://oxygen.lcs.mit.edu/.

[6] C. Becker et al, BASE - A Micro-broker-based Middleware For Pervasive Computing, *in Proc 1st IEEE International Conference on Pervasive Computing and Communication* (PerCom03), Fort Worth, USA, 2003.

[7] E. Christopoulou and A. Kameas, GAS Ontology: an ontology for collaboration among ubiquitous computing devices, *International Journal of Human – Computer Studies: special issue on Protégé* (to appear).

[8] W. Gaver, Technology Affordances, *Proc. ACM CHI'91 Conference*, 79–84.

[9] J. J. Gibson, *The Mifflin*, 1979.

[10] T. R. G. Green and M. Petre, Usability analysis of visual programming environments: a cognitive dimensions framework, *J. Visual Languages and Computing*, **7** (1996), 131-174.

[11] J. Habermas, *The theory of communicative action I and II* (trans. T. McCarthy), 1984.

[12] J. Hill et al., System architecture directions for networked sensors, in *Architectural Support for Programming Languages and Operating Systems* (2000) 93-104.

[13] L. E. Holmquist, F. Mattern, B. Schiele, P. Alahuhta, M. Beigl and H.W. Gellersen, Smart-Its Friends: A Technique for to Easily Establish Connections between Smart Artifacts, in *Proc UBICOMP 2001*, Atlanta, GA, USA, Sept. 2001.

[14] A. Kameas, D. Ringas, I. Mavrommati and P. Wason, eComP: an Architecture that Supports P2P Networking Among Ubiquitous Computing Devices, in *Proc. IEEE P2P 2002 Conference*, Linkoping, Sweden, Sept. 2002.

[15] A. Kameas, et al, An Architecture that Treats Everyday Objects as Communicating Tangible Components, in *Proc. 1st IEEE International Conference on Pervasive Computing and Communications* (PerCom03), Fort Worth, USA, 2003.

[16] I. Mavrommati, P. Markopoulos, J. Calemis, and A. Kameas, Experiencing Extrovert Gadgets, in *Proc. HCI 2003*, Vol. 2, Research Press International, 179-182.

[17] I. Mavrommati, P. Markopoulos and A. Kameas, Visibility and accessibility of a component-based approach for Ubiquitous Computing applications: the e-Gadgets case, in *Proc. HCI 2003*, Vol. 2, Research Press International.

[18] I. Mavrommati and A. Kameas, End user programming tools in ubiquitous computing applications, in *Proc. HCI 2003*, Vol. 2, Research Press International.

[19] I. Mavrommati, A. Kameas and P. Markopoulos, An editing tool that manages the device associations, in *Proc. 2AD conference on Appliance Design*, Bristol, May 2004 (to appear).

[20] M. McLuhan, The Medium is the message, Bantam books, New York, 1979.

[21] T. D. Meijler and O. Nierstrasz, Beyond objects: components, in M. Papazoglou and G. Schlageter (Eds) *Cooperative information systems: current trends and directions*, Academic Press, 1997

[22] D. A. Norman, *The Psychology of Everyday Things*, New York, Basic books, 1988

[23] D. A. Norman, *The Invisible Computer*, MIT press, 1998.

[24] M. Román and R.H. Campbell, GAIA: Enabling Active Spaces, in *Proc 9th ACM SIGOPS* European Workshop, Kolding, Denmark, September 2000, 229-234.
[25] J. G. Schneider and O. Nierstrasz , Components, scripts and glue, in J. Hall and P. Hall (Eds) Software architectures – advances and applications , Springer-Verlag, 1999, 13-25
[26] H. Thimbleby, Affordance and Symmetry, in *Proc DSV-IS 2001* C. Johnson (Ed), LNCS2220, Springer Verlag Berlin Heidelberg, 199-217.
[27] B. Warneke et al., Smart Dust: Communicating with a Cubic-Millimeter Computer, *IEEE Computer Magazine,* Jan 2001, 44-51.