UNIVERSITY OF SOUTHAMPTON
Faculty of Engineering, Science and Mathematics
School of Electronics and Computer Science

A progress report submitted for continuation towards a PhD

Supervisor: Prof. Wendy Hall
Examiner: Dr. Terry Payne

Evolving Smart Environments

by Ben Deitch

December 17, 2004

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

A progress report submitted for continuation towards a PhD

Evolving Smart Environments

by Ben Deitch

Current research in the field of smart environments includes the assumption of the presence of infrastructure from the smart environment's inception. However, due to economic constraints, it is likely that many smart environments will instead have to evolve as the devices forming their infrastructure are slowly introduced over a period of time. In this document the problem area of evolving smart environments is discussed, together with background work in pervasive computing and some related areas. Two initial experiments, more generally related to the area of smart environments, are then described. Finally, an approach based on semantic service composition and substitution, intended to answer some of the issues involved in enabling an evolving smart environment, is presented together with a plan for future work including an experiment to test this approach.

Contents

Chapter 1

Introduction

Current research in pervasive computing deals largely with the area of smart environments. A smart environment is a region of physical space that is instrumented with computing technology, such as sensors and actuators, which is intended to provide non-intrusive support to the occupants of the environment as they work to accomplish their tasks.

Research in this area is consistently based around the assumption that the technology infrastructure forming a smart environment is installed in its entirety before the system is used.

A more realistic approach accepts that due to various constraints, not least economic constraints, the adoption of pervasive computing and smart environment technology will likely be piecemeal and distributed over significant time. In such cases there will not be a smart environment immediately; items of pervasive computing technology, or smart devices, will gradually be introduced into a physical space over time.

In order to fulfill the promise of pervasive computing, and to attempt to provide a cohesive smart environment, these devices will have to automatically cooperate, possibly across different generations of technology, to provide functionality that makes best use of all their abilities in combination.

However, it is not enough for devices to simply reconfigure to incorporate the new functionality each time another device is added; Users will have become used to the behaviour of their current combined devices; they may well have spent time adjusting the behaviour of devices to their preferences.

Therefore it is necessary for the current configuration to inform the automatic creation of the new configuration. Thus, each configuration of devices can be seen as a step to the next configuration produced by the addition of a new device. In this way the system formed of the combination of devices can be said to be ‘evolving’; in the remainder of this document this type of system is referred to as an ’Evolving Smart Environment’ or ESE.

1.1 Research Aims

There are a number of non-trivial research challenges that must be met in order to enable us to build ESEs. These challenges are presented below, formulated as a number of questions; followed by a more focused discussion on the specific problem of evolving functionality.

1. How can an ESE recognize the features of new smart devices?
2. How can an ESE utilize the features of new smart devices to provide combined functionality?
3. How can new smart devices be automatically configured as they are added to the ESE?
4. How can an ESE handle the addition and removal of smart devices gracefully without failure? (i.e. how can an ESE devolve as well as evolve?)
5. How can the behaviour of the ESE be predictable to the non-expert user as new smart devices are added and removed?
6. How can smart devices added over a long time period, with possibly significant variations in technical sophistication, interoperate effectively?

Clearly, considering a number of these challenges is beyond the scope of this document (and beyond the scope of a single PhD). It is therefore necessary to focus on a more specific problem within this area: functional evolution.

When considering the functional evolution of a smart environment it is useful to model it simply. We consider the environment to be formed of two sets: the set of smart devices that form the environment, and the set of tasks that can be achieved using those devices. It is important to note that this set of tasks includes both the tasks that the occupants might wish to accomplish and also tasks that the environment itself might be doing automatically.

We thus describe the functional evolution of a smart environment via the set of tasks achievable in that environment. If new tasks are added to the set then we consider that environment to have evolved. Similarly (and perhaps more significantly) if some of the tasks already available are improved in some way then the environment is, again, considered to have evolved. It is important to note that, although this functional evolution is envisioned as occurring mainly when new devices are added, the smart environment may improve its task set independently of changes in the set of devices.

There are a number of deeper research issues raised on consideration of this model of an evolving smart environment:

There is clearly a difference between descriptions of tasks that occupants of a smart environment may wish to accomplish and the actual atomic functions or services provided by the devices in that environment. A way needs to be found to map from an abstract high-level description of a task down to the actual device functions that must be used.

We describe the evolution of a smart environment in terms of the set of tasks increasing or elements of the set improving in some way. This description poses two further problems: Firstly, how should the set be increased? And secondly, how can tasks already present be improved?

The problem of providing new composite functionality from a set of devices is itself a challenge. This new functionality can perhaps be classed as coming from two distinct sources: A source external to the ESE or a source within the ESE.

Sources external to the ESE could be such things as an external repository of task plans, perhaps provided by the device manufacturers themselves. Other sources could be shared repositories populated by the users themselves; as they form their own new task compositions these could then be shared with a user community. Then there is also the fact that the occupants of the ESE would want to create there own private task compositions. The ability to share tasks and provide generic plans is also reliant on a mapping between abstract high-level descriptions of tasks and the actual functions of the devices. Other issues involved in allowing users to create their own task compositions are related to ease-of-use for the users; what kind of interfaces are needed to allow the comprehension of the possible task compositions from the point of view of unskilled users?

Internal sources for new task compositions come from the ESE itself. These could be something as naive as random concatenation of compatible tasks in order to generate new tasks which would then be filtered, either by the ESE or perhaps by its occupants. Alternatively, more advanced techniques based on machine learning could provide a more direct evolution of functionality based on things such as user modeling.

Looking at the problem of improving tasks already extant in the task set leads to three further questions:

How should tasks be described so that they can be improved upon? Not only do the tasks need to be described in some human readable form, in order that users can understand what functionality is being offered by the environment, but if the tasks are to be improved in an automatic process then an appropriate description is needed to facilitate this process.

How can we define what is an improvement? Very specific criteria would need to exist to determine if a task has been improved rather than simply changed. These criteria could be based on low abstraction level considerations such as resource usage levels and exception handling robustness. Perhaps more useful criteria would come from higher abstraction level considerations, for example: requiring less control from users in order to complete the task, or providing better adaptation to general user preferences.

How can these criteria then be used to guide the improvement process? Once criteria for determining if a task has been improved have been established, it is then necessary to provide a way for these criteria to guide the improvement process. This in turn can be seen to guide the evolution of the ESE to some extent. Perhaps if the occupants of the ESE can prioritize or otherwise rank the improvement criteria they can control the evolutionary process in a more understandable manner.

The author believes this problem area to be addressable using an approach based on semantic web services and task composition and substitution. The details of this approach are presented in the Future Work section of this document.

Chapter 2

Background Work

2.1 Pervasive Computing

The field of Pervasive Computing has not yet been defined conclusively in that there is not a commonly agreed-upon, comprehensive definition that could be quoted here. This lack of a definitive description of the field is reflected in the fact that it is still referred to in the literature under different titles, such as Ambient Computing and Ubiquitous Computing. This document is not intended to join the debate as to whether or not these terms are synonymous and is thereby restricted to using the term Pervasive Computing (PerComp) which is defined within this document as:

> *The use of a number of computing devices, usually embedded in the environment and interconnected via one or more networks, to provide services to users in a non-intrusive manner.*

Further to this definition there are some basic characteristics of PerComp which, while perhaps not definitive of the field, give an idea of the many challenges involved:

- Abundance

  PerComp potentially deals with very large numbers of devices. One example is Smart Dust (Kahn *et al.* 2000) which refers to the use of sensing devices eventually intended to be so small, inexpensive and numerous that they can be scattered in an environment like dust. Users are another potentially abundant factor in PerComp systems. For example, it is quite conceivable that a PerComp service environment in an airport terminal would need to be able to deal with large numbers of people making concurrent demands on its resources. Further to this, where a PerComp system is attempting to provide services to a large number of users, there could well be an abundance of data as the system might have to process high volumes of information associated with each of the users.

- Heterogeneity

  Perhaps one of the more interesting challenges in PerComp is handling the heterogeneity that can be present in many aspects of a PerComp system. Devices involved can range from small, simple sensor nodes up to large server systems. The networks that may be utilised can similarly range from short range, ad-hoc radio communication (e.g. Bluetooth) to global networks such as the Internet. Whilst these levels of heterogeneity may be found in individual PerComp systems, even greater differences can be experienced by mobile devices traveling through multiple PerComp environments, in order to provide consistent levels of service to the user such mobile devices would have to handle widely varying levels of resource availability and resource types in each environment.

- Intelligence

  Intelligent systems, using tools borrowed from AI, can be found in PerComp in many areas and in a variety of levels of complexity. An important example would be the processing of contextual information using rule-based approaches originally found in expert systems. This is significant as many of the features of pervasive computing systems (such as tailoring information and interaction requests to a user's current location and task context) rely on such context processing, although rule-based approaches are not the only option in this case. Another area in PerComp that can gain from advanced autonomic behaviour is system self-configuration. Problems considered in distributed AI (such as Agent-based Communities) have much in common with some PerComp systems such as those that involve a large, changing number of distinct components that should interoperate automatically.

- Environment-based Applications

  Unlike most examples of current computing where the activities of the users are centred around the computers themselves, in PerComp the focus is on the user and their environment. From the definition of PerComp given above, where it is stated that devices are usually embedded in the environment, it follows that such devices provide services relevant to that environment (otherwise there is little reason to embed them there in the first place). One environment that can be considered an example of PerComp is the modern car. In a reasonably advanced modern car a combination of sensors, computing power and carefully designed user interfaces provide services of information and control to the user in a non-intrusive manner intended to aid the main task (driving the car). Similarly, we can see the potential for other environments to be augmented in order to aid users in performing their tasks, perhaps especially those tasks specific to a certain environment. An example could be a smart meeting room. Such a meeting room might provide useful information such as the expected arrival time of delayed participants or notification of external events that are relevant to the topic of the meeting. Support for collaboration amongst the participants such as a distributed white board or automatic note taking facilities might also be present. These applications are based within the environment improving or adding to the facilities in that environment. Such environments are often referred to as 'Smart Environments' or 'Smart Spaces' (Coen 1998).

2.2 Agent-based Computing

2.2.1 Definition

Wooldridge (2002) presents the following definition of an agent:

> "An *agent* is a computer system that is *situated* in some *environment*, and that is capable of *autonomous action* in this environment in order to meet its design objectives."[1]

There are two important points to note from this definition. Firstly, the fact that an agent is "situated in some environment" indicates that the agent can sense, to some greater or lesser extent, its environmental conditions (which can be seen as part of its context). A further implication is that the agent may well have the ability to effect changes in its environment, suggesting the possibility of a two-way

[1]emphasis in original

relationship between an agent and its environment. The second important point is that an agent is capable of autonomous action. Agents are often implemented with some kind of reasoning mechanism, in fact the field began with research into distributed artificial intelligence systems. However it is not required that an agent make use of AI style reasoning to be called an agent. Agents that do utilize some form of symbolic or other high-level reasoning are said to be ‘strong’ agents, whereas those agents that do not make explicit use of AI techniques are called ‘weak’ agents (Dale & Mamdani 2001).

As an addendum to Wooldridge’s definition of an agent, many researchers consider a key element in the definition of an agent to be that it operates under a distinct thread of control. This is an important factor when considering the distributed systems constructed from multiple agents, referred to as ‘Multi-agent Systems’.

2.2.2 Agent-oriented engineering

When multi-agent system solutions are applied to problems the practice is referred to as agent oriented software engineering. Jennings (2001) talks of several key characteristics of the agent oriented paradigm. The suitability of the agent paradigm for use in pervasive computing can be discussed by considering each characteristic in turn:

- Organization Models

  The individual agents in a multi-agent system can be organized in a wide variety of ways. Standard distributed computing models can be applied (such as the Master-Slave or Task Graph models) as well as more complicated organizations that rely on the fact that each agent is autonomous. More importantly; organizations in multi-agent systems can be dynamic. This means that they might be created and dispersed automatically by the agents as the need arises. This ability to dynamically reconfigure has obvious applications in pervasive computing environments. The more advanced organisational models provide a way to model pervasive environments where systems with different priorities will need to work together to provide a common service infrastructure.

- Distributed Computation

  There are two ways in which the distributed computation model presented by agent-based computing is particularly suited to pervasive computing:

first, it is decentralized with multiple loci of control. This is useful when trying to answer the need in pervasive computing for multiple devices or subsystems to be responsible for themselves yet still able to interoperate effectively. Secondly, the communication models used in agent-based computing are usually specified at a high level of abstraction (Moreau 2002). This allows communication to pass across different technologies more easily which would obviously be a boon in pervasive computing where networks will be formed from many and varied sytems.

- Encapsulation

  As with object-oriented engineering, encapsulation figures largely in the field of agent-oriented engineering. The reduction in engineering complexity that this offers is not unique, however the advantage that the agent approach brings is the ability to intelligently decide at runtime what data to expose to which consumers. This is useful when dealing with privacy in the field of pervasive computing. Consider the example of a user moving through a public smart space. The various services offered by the space require some knowledge of the user and his preferences, however the user wants to control access to his personal profile data and reveal only as much as is needed and only if the benefit is sufficient to justify the access. This type of scenario is particularly suited to agent-based computing.

- Autonomy

  Agent-based computing offers a range of autonomic behavior. Multi-agent systems can include simple auto-configuration and advanced artificial intelligence mechanisms. The autonomic models possible with such systems are both flexible and powerful. In pervasive computing there is a need for more than simple auto-configuration and self-management. The level of complexity that autonomic elements of a pervasive system must deal with is increased by a number of factors: The lifetime of components in a pervasive system may well be considerably longer than components in current systems which will be replaced more often. Physical access is not guaranteed and therefore systems must deal with cumulative failures over time, gracefully providing useful functionality for as long as possible. Due to the progressively interconnected nature of pervasive systems as new systems are added, each system must be able to handle changing environments that were not foreseen at design time. In fact, the basic nature of the system as a whole may go through many changes during the lifetime of a component. This component must be able to adapt to best service these changing needs. Agent-based computing offers a cohesive approach to these complex demands.

- Semantically Aware

  Agents must understand each other in order to interoperate. The approach taken to facilitate such operation usually involves describing data ontologically i.e. providing a common semantic framework that agents can use to communicate the meaning of pieces of information. Pervasive computing shares this need for the communication of data together with its semantic meaning. Therefore, approaches already explored in agent-based computing can be usefully applied.

In summary, the agent-oriented engineering paradigm offers many benefits and interesting approaches to the problems that must be solved in pervasive computing.

2.3 Semantic Technologies

Semantic technologies seek to bridge the gap between human understanding of data, i.e. semantics, and machine understanding of data, i.e. syntax. Helping a computer to ‘understand’ the information it is processing is a problem that is present in many areas; from classical artificial intelligence to mobile computing and the internet.

> “Most of the Web’s content today is designed for humans to read, not for computer programs to manipulate meaningfully. Computers can adeptly parse Web pages for layout and routine processing — here a header, there a link to another page — but in general, computers have no reliable way to process the semantics: this is the home page of the Hartman and Strauss Physio Clinic, this link goes to Dr. Hartman’s curriculum vitae.” (Berners-Lee *et al.* 2001)

While much of the work on semantic technologies is focused around the Semantic Web (Berners-Lee *et al.* 2001), the need to specify semantics using a generic approach is also shared by other research areas in computer science. In the past if different systems needed to share data or tasks then the relevant semantics of the data needed to be defined and shared *a priori*. The way in which such semantics was defined was often informal and mostly application specific.

Semantic technologies offer the ability for the semantics of data to be formally defined in such a way that the data can be easily shared together with its meaning. This does more than allow us to easily design systems that interoperate, it introduces the potential for the dynamic introduction of new semantics to a system at runtime.

2.3.1 Semantic web languages

A number of languages figure largely in the field, each was designed primarily for use with the semantic web but now they are becoming increasingly useful in other areas:

- *RDF (Resource Description Framework)*

  This language is defined in a W3C recommendation. The W3C provide a primer document that is regularly updated to reflect the latest version of RDF. In the introduction to this document the following description is given for RDF:

  > "The Resource Description Framework (RDF) is a language for representing information about resources in the World Wide Web. It is particularly intended for representing metadata about Web resources, such as the title, author, and modification date of a Web page, copyright and licensing information about a Web document, or the availability schedule for some shared resource. However, by generalizing the concept of a "Web resource", RDF can also be used to represent information about things that can be identified on the Web, even when they cannot be directly retrieved on the Web. Examples include information about items available from on-line shopping facilities (e.g., information about specifications, prices, and availability), or the description of a Web user's preferences for information delivery." (W3C 2004*b*)

  An important point to note in this description is the fact that RDF can be used to describe resources that are not directly part of the web. These resources may not even be digital in nature, RDF could just as easily be used to describe a pet cat as it could be used to describe a piece of website content.

- *DAML+OIL (DARPA Agent Markup Language, Ontology Inference Layer)*

  RDF is used to describe resources however it is not advanced enough on its own to describe the meaning of information sufficiently formally or in such a way that reasoning engines can be applied. Some form of ontology language is necessary. Here we use the word 'ontology' to refer to a set of building blocks used to form a conceptual representation of a knowledge domain. DAML+OIL is one such language.

  DAML+OIL evolved from a merger of DAML-ONT (an earlier DAML ontology language) and a description logic called the Ontology Inference Layer (OIL). The two main aims of DAML+OIL were to provide an easily comprehensible ontology language that was backwards compatible with current web languages, and to ensure that the language had a strong formal underpinning. (McGuinness *et al.* 2002)

  To achieve the first stated aim DAML+OIL was defined in RDF/XML (a version of RDF that is serialized into XML). The second aim was accomplished by using formal description logic as a base for the language design, as well as providing an equivalence-preserving translation from DAML+OIL into first order logic.

- *OWL (Web Ontology Language)*

  OWL is also defined in a W3C recommendation. In the language overview the following description is given:

  > "The OWL Web Ontology Language is designed for use by applications that need to process the content of information instead of just presenting information to humans. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with a formal semantics. OWL has three increasingly-expressive sublanguages: OWL Lite, OWL DL, and OWL Full." (W3C 2004*a*)

  DAML+OIL was used as the starting point for OWL, which is now generally considered to have superseded DAML+OIL. OWL is also expressed in RDF/XML. As mentioned in the W3C's description, OWL has three sublanguages. Going from OWL Lite to OWL Full, each language gains in expressiveness but becomes harder for automated reasoners to process.

2.3.2 The semantic web in pervasive computing

As stated earlier, the semantic web is the driving focus behind these languages and the automated reasoning tools that accompany them. However the ability to machine-process the meaning of data is a boon to many other fields as well. There are some compelling reasons to make use of ontology languages in pervasive computing.

Ontology languages offer benefits to pervasive computing systems mainly in the area of interoperability. The advantages of a semantic approach can be considered under the following headings:

- Data Utility

  Pervasive computing shares the advantage that is common to any field that makes use of semantic technologies: data utility. What this refers to is the increase in the usefulness of data when its meaning can be communicated in machine readable form as easily as the data itself.

- Standards

  As with data utility, the benefit of shared standards is not unique to pervasive computing. However it is important to note that standards can be a double-edged sword. In a relatively new field such as pervasive computing, solidification via standards can restrict the development of new ideas and new approaches. Whilst the W3C has settled on one main ontological language (OWL) and its underlying fabric (RDF) there is still a great deal of openness within these confines. Inherent in OWL, and even semantic technology as a whole, is the idea that ontologies should be easily extensible without necessary alteration to the originals. The use of Uniform Resource Identifiers (URI's) throughout these languages enables easy cross-utilization of ontologies and concepts contained therein. Thus far a consensus approach has been used by the semantic community to agree upon certain 'base' ontologies that express core concepts such as time and space. This enables a much faster pace of development than if such ontologies were fixed in the standards process, although the consensus approach is clearly open to abuses and potential conflicts.

  Hendler (2001) suggests that the reasoning and logic engines associated with semantic technologies should be considered distinct from the ontologies themselves. Even so, the fact that the reasoning is over common ontological structures provides the possibility that standards can be found for the processing of semantic data as well as its representation. Such standards would benefit pervasive computing by moving towards a common toolkit for the construction of pervasive system components.

- Semantic Service Description

  Semantic descriptions of services and devices allow more advanced discovery that could be essential for the flexible and dynamic nature of pervasive systems. A language that provides this and more is the Ontology Web Language for Services (OWL-S) (Martin *et al.* 2004). OWL-S offers a semantic description layer for web services that is not provided with the more low-level service-syntax oriented Web Services Description Language (WSDL). It is part of a body of work that attempts to develop languages and architectures to provide richer semantic specification of web services. This body of work is referred to as Semantic Web Services.

  > "OWL-S (formerly DAML-S) is an OWL ontology with three interrelated subontologies, known as the profile, process model, and grounding. In brief, the profile is used to express what a service does, for purposes of advertising, constructing service requests, and matchmaking; the process model describes how it works, to enable invocation, enactment, composition, monitoring and recovery; and the grounding maps the constructs of the process model onto detailed specifications of message formats, protocols, and so forth (normally expressed in WSDL)." (Martin *et al.* 2004)

  An important point to note from the description above is that OWL-S provides a grounding to map the constructs of the process model onto WSDL. This is particularly interesting as it essentially means that OWL-S can provide a semantic wrapper around the web service method invocations. Whilst this would obviously incur overheads in addition to those normally found with web services, it offers the potential for designing distributed systems at the abstraction level of semantics rather than technical functionality. The inherent flexibility, extensibility and possible autonomy in this model indicate it is worth exploring for possible applications in pervasive computing. Further to this, current research into automatic service composition goes toward providing an intelligent adaptive infrastructure for pervasive systems.

Semantic technologies have already played a key role in pervasive computing projects. Some examples follow:

In the University of Maryland, Baltimore, Chen *et al.* (2003) have been building a context broker architecture (CoBrA) for smart spaces that utilizes ontologies. This design is based on a central agent that is responsible for gathering, processing and sharing context information with the other elements of the system.

The broker system addresses bottleneck issues by replacing a single central context broker with a 'broker federation' in large-scale cases, where each broker is responsible for subsections of an overall space (such as buildings in a university campus). However there is still an issue with a single broker agent being responsible for all context in a certain area. Whilst the broker federation does reduce bottlenecks it does not guarantee scalability, more importantly the broker agent must be assumed to be trustworthy enough to be allowed access to all relevant context information. A peer-to-peer approach where users' agents are responsible for sharing and coordinating context information may offer solutions to these problems.

The (CoBrA) system relies heavily on ontologies, "helping the context broker to share contextual knowledge with other agents and enabling it to reason about context" (Chen *et al.* 2003). The ontologies were defined in OWL. Whilst this use of ontologies does go some way to providing an open system it still suffers from the fact that the ontologies used are highly specific to CoBrA and its use cases. Adding new components to this system might therefore be easier than if the context information was presented as Java classes (for example) but there is still a requirement that components be built specifically for CoBrA. However this lack of generality is recognized by the authors: "It may be necessary to re-organize the class hierarchy if the ontology is reused to support a different context-aware application." (Chen *et al.* 2003) In their discussion of future work the authors state their intention to move to consensus ontologies such as DAML-Time and DAML-Space.

What is most interesting about the CoBrA project is the development of a prototype ontological inference engine called F-OWL, implemented using Flora-2, an object-oriented knowledge base language and application development platform. The authors assert that the benefits of F-OWL include more efficient execution through its use of cache tables of previous results and a syntax more in common with OWL due to its object-oriented design.

The MyCampus project at Carnegie Mellon University (Sadeh *et al.* 2002) also relies heavily on semantic technologies. The authors state: “The power and scalability of the environment directly derives from a set of ontologies for describing contextual attributes, user preferences and web services, making it possible to easily accommodate new task-specific agents and new Web services.” Their work builds directly on the DAML-S work previously done at Carnegie Mellon University. (Ankolekar *et al.* 2002)

Chapter 3

Investigatory Prototypes

3.1 Managing Intrusiveness in a Meeting Room Scenario

Pervasive computing devices are becoming increasingly common. Devices like laptops, mobile phones, video phones, smart white-boards and public displays are making it possible for people to be contacted easily and receive notifications in more situations than ever before. These interruptions can impact on both individual and group tasks.

Having said that interruptions have an impact it is important to note that not all interruptions are 'bad' i.e. harmful to the task at hand. Interruptions can be classified as either *intrusive* or *task support*. (Ramchurn *et al.* 2004)

Intrusive interruptions are those where the user or the group has their stream of consciousness forced from one focus to another, whereas an interruption that supports the current task requires less of a focus shift and adds new helpful information. Obviously a message is not just simply intrusive or not, but can in fact have varying levels of intrusiveness depending on the context of the recipient. Determining the intrusiveness of a message involves the preferences of the user and also, if that user is part of a group, the preferences of the group. This last point is intuitively evident in cases where the receiving device has the potential to disturb the entire groups focus. Thus, when designing a system to manage these interruptions, it is necessary to consider the more complex problem of reconciling an individuals preferences with the preferences of the group.

Agent-oriented engineering offers an ideal model for dealing with what is essentially a distributed problem involving possibly conflicting interests. In this experiment we explore this problem via the scenario of a meeting in a smart meeting room. This meeting room contains tools such as a shared public display and a wireless network. Each user is represented by an agent that is aware of their individual preferences. The interests of the group are represented by a server agent responsible for that meeting room. The server agent coordinates the display of any incoming messages to those users in the meeting room. A cost is associated with each device that the message may be displayed on. This cost is proportional to the intrusiveness of that device. For example, an email client on a user's laptop that silently shows a text notification is much less intrusive (and thereby cheaper) than a mobile phone whose ringing could well disturb the entire group. In order to display a message a user's agent must pay the cost to the server agent from an initial budget set by the server agent at the beginning of the meeting.

A user's preferences are recorded in their agent as a utility function: $U(Sender,\ Subject) \rightarrow u$. This function determines the utility returned to an agent's budget upon displaying a message that satisfies these preferences. Each agent seeks to maximise its utility. In order to represent the interests of the group as well as those of its owner, an agent must negotiate with the other users' agents prior to paying for the display of the message. The agents negotiate on the subject and sender of the message. Where more than one agent has an interest in displaying the message (i.e. would receive some utility) they can combine their budgets in order to display the message on a more expensive device (such as the public display) which would be accessible to all interested users.

The preferences of the group are also represented through the use of an 'intrusiveness dial' which allows the users to scale the cost of devices according to the state of the meeting. During a time in the meeting when the group is deep in discussion this dial would be turned up to increase the cost of displaying messages, thereby reducing the number of interruptions. During a break in the meeting, the dial could then be turned all the way down, allowing all messages (including previously queued messages) to be received.

In order to demonstrate the feasibility of this approach a simulation was constructed to test the hypothesis that the intrusiveness of incoming messages could be reduced.

3.1.1 Implementation

The architecture of the simulation implementation is shown in figure 3.1.

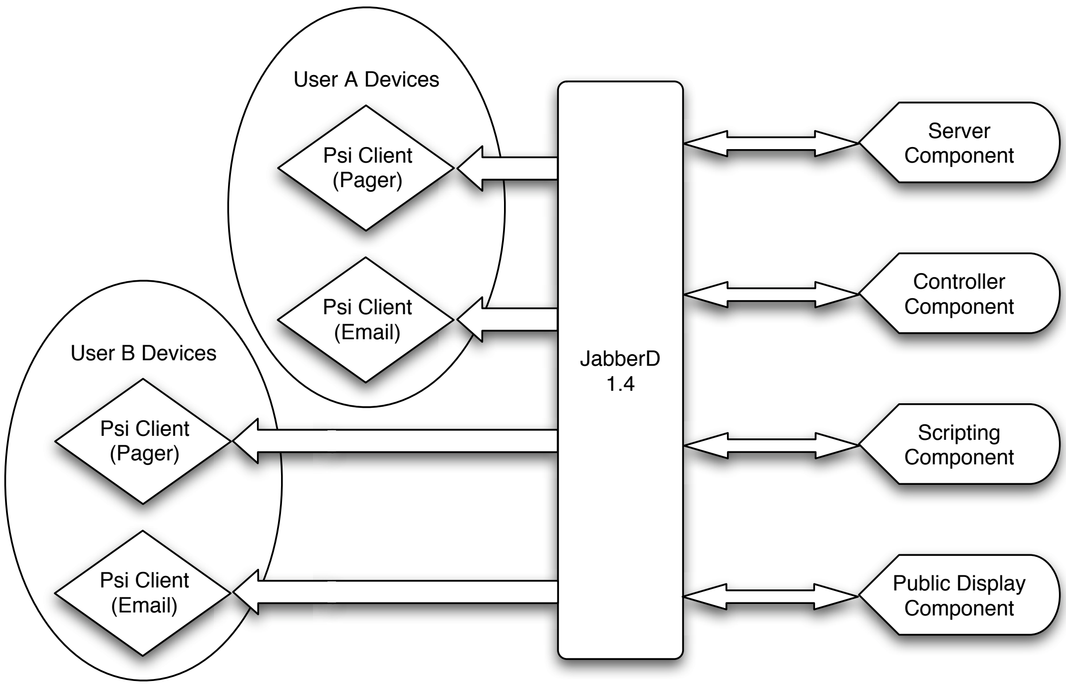

FIGURE 3.1: Architecture of meeting room system simulation

The system was built on the Jabber platform (JSF 2004). Jabber is an open-source instant messaging system that is based on XML messages and the Extensible Messaging and Presence Protocol (XMPP) (IETF 2004). At the core of the system was the JabberD Jabber server (JabberD 2004) which was used to route XML packets between the other components.

The Server component housed all the agents. This component was responsible for performing the agent negotiation and thereby determining the destination for each re-routed message.

The Controller component presented the interface between the users and the system. The representation of the intrusiveness dial is part of the GUI presented by this component.

The Scripting component was a tool used for the evaluation of this system. This component simulated incoming messages from a variety of sources sent to a range of users' devices.

The Public Display component simply displayed any messages that it received on a publicly visible screen.

Finally each users devices were represented by Jabber clients. The Jabber client, “Psi” (Psi 2004), was chosen because it is highly configurable and could be set to represent less intrusive devices (like an email client) and more intrusive devices (such as a pager), by adjusting the audio and visual notifications given for incoming messages.

3.1.2 Results

The evaluation of the experiment was based on a comparison against a baseline of using no interruption management system. Our hypothesis was that this system would successfully reduce the number of intrusions while still allowing messages to be received that were either of sufficient importance to a user or relevant to the task at hand. In successive execution runs of the simulation, we found this to be the case. Therefore, the experiment was successful in that it demonstrated how a multi-agent system based approach could offer a solution to this problem.

The system as implemented did suffer from two important drawbacks related to privacy. Firstly, when the agents were negotiating, the full content of a message would be given. This is obviously far from ideal. Secondly, should an incoming message be highly valued by a number of agents then it would almost certainly be shown on the public display. No accounting is made for a users privacy concerns over the contents of such a message.

An obvious area for future work related to this experiment would be to implement a working system instead of just a simulation. Then, such issues as scalability and decidability could more realistically be explored.

3.2 Laboratory-based User Location Monitoring System

This experiment was designed to explore the issues involved in building a system that would allow new components, which operated in radically different ways from those components that were already part of the system, to be added without requiring any changes to the existing components. This aim derives directly from the conditions for an evolving smart home stated in section 1.4 “Research Goals”.

It was hypothesized that using a semantic language for inter-component communication could allow new kinds of components to be integrated easily as long as some common semantic base existed between the components. The scenario chosen to provide a focus for the development of the components was a laboratory-based user location monitoring system. There is a variety of approaches to the problem of acquiring location information for users inside a building. Examples range from sensing user location via a transceiver badge to inferring location from calendar appointments. This scenario thereby provided a choice of possible components, i.e. locating methods, that could be developed for the experiment.

3.2.1 Implementation

For the sake of simplicity the infrastructure was based on the Elvin message passing system. Elvin provides efficient content-based message routing via a system of producers and consumers that seemed ideal for the purpose of the experiment. It was decided that a Bluetooth-based sensing system should be the initial component. Bluetooth technology is both cheap and well understood and Bluetooth-enabled personal devices (to act as the transponders) are relatively common.

The sensing system is based on a number of Bluetooth-enabled computers which detect the presence of other mobile devices that are Bluetooth-enabled and then report this information as Elvin notifications. This has the advantage of using currently existing computers with Bluetooth capabilities rather than building or purchasing new sensing devices.

In order to allow reuse on as many operating system platforms as possible, the sensing code was written in JAVA using the JSR-82 Bluetooth API specification (JCP 2004). This specification is fairly recent and the number of available implementations is limited. There is currently no fully working implementation for the Apple Macintosh platform and Windows implementations that are available are limited to a subset of Bluetooth chipsets. However, a reliable implementation is available for Linux. It is assumed that new versions of the API will be released on the other platforms.

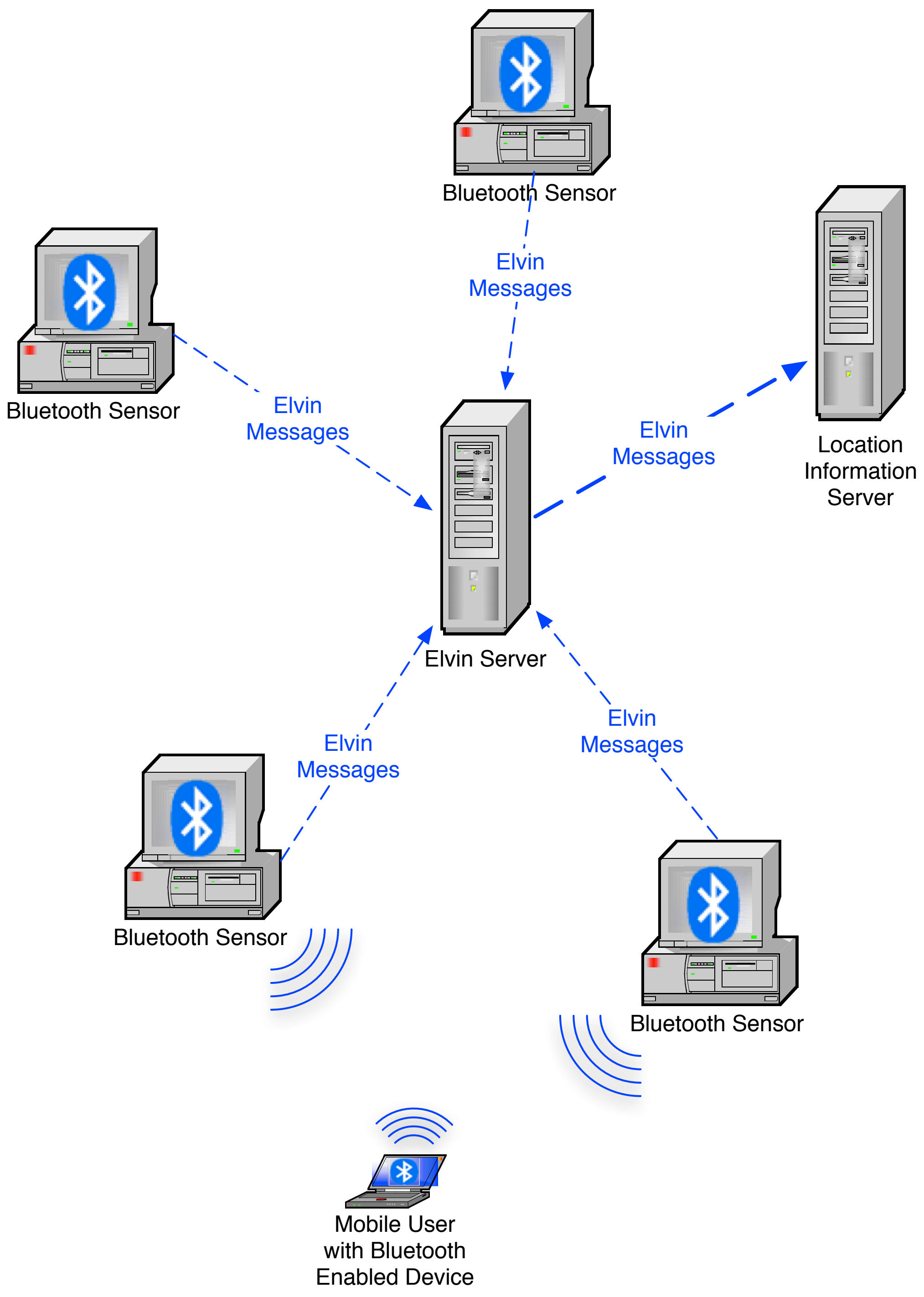

FIGURE 3.2: Architecture of bluetooth-based location monitoring system

3.2.2 Results

Currently the experiment is still under development. The initial attempt to build the experiment as a practical implementation of a location sensing system has perhaps been a mistake, much time has been taken up with the minutiae of engineering such a system. This has drawn the focus of work away from the essential task of testing the hypothesis. In order to redress this issue it has been decided that the experimentation will continue via a model of a smart environment. Modeling the environment, the devices and the users within the environment, will allow both a wider range and better control of experimental variables. Importantly, this approach will also lead to more time being spent on the aspects of device interoperability and integration rather than the practical engineering problems involved in building an actual implementation. The step of implementing a real system could follow once results have been acquired from the experimental model.

Chapter 4

Conclusions

The majority of current research that deals with smart environments operates on the assumption that the main components of the environment, such as communication infrastructure, sensors and user interface devices, are fairly unchanged during the lifetime of the environment, or at least the lifetime of any experiment. However, there are a number of reasons why this approach to developing pervasive computing environments is not entirely suitable. Perhaps the most important is that there is a significant economic cost involved in installing the equipment for a smart environment, which many users (both domestic and commercial) may find difficult to justify. Users may also be uncomfortable with the changes involved in moving from an environment with no technology to one which is fully 'kitted out' with smart devices. This may seem like too radical a change for some. Also, in environments such as a domestic home, many smart devices are extended versions of standard devices, an example might be the internet-enabled fridge. In such a case it is intuitively obvious that users are not likely to simply replace all their home equipment with new, smart versions. It is much more likely that devices will gradually be replaced as they grow old or perhaps as the home owners are given new devices, as presents for example. In general, the majority of smart environments are likely to gradually evolve from low-tech environments as economic and usability constraints allow.

It is assumed then, that smart devices are often likely to be introduced into environments individually or in small groups. These devices will perhaps provide their own user interfaces and self-contained smart functionality rather than rely on the presence of other smart devices. There is a danger in this scenario, as more devices are introduced that may or may not integrate with some subset of the devices already present in the environment, that 'islands' of smart functionality could develop. Rather than be able to interact with the environment as one cohesive entity, users would have to deal with each smart island separately. This has two major drawbacks: Firstly, usability suffers greatly as users are forced to learn multiple interfaces and perhaps deal with several different interface paradigms. Secondly, as the functionality of each smart island is distinct from the others there may be redundant duplication and most importantly, opportunities to provide more comprehensive and intelligent functionality would be lost due to the lack of integration.

This problem can be considered from two distinct perspectives. From the user's perspective the problem is one of providing a coherent common interface and integrated behaviour. From the device perspective the problem is one of interoperability and cooperation. These problems are exacerbated by the need to provide a solution that deals gracefully with occasional but significant changes over time in the device membership of the smart environment. It is not enough just to provide a user with a coherent interface and integrated behaviour but also, as new devices are added, the possible evolution of both the interface and the entire system's behaviour should be consistent and predictable to the user.

The first experiment described in this document, a multi-agent approach to managing intrusiveness, demonstrated that a distributed AI model can be successfully used to manage and coordinate multiple devices. This would suggest that agent-based computing may be a useful way of tackling the problem of presenting a coherent common interface across the devices in a smart environment.

The second experiment is still in its initial stages, the bluetooth detection code written thus far will be integrated into the experiment as described in the next section "Future Work".

4.1 Future Work

My future work aims to move towards a solution to the problem, stated in the introduction, concerning evolving functionality via task improvement.

Beginning with the simple model of an ESE as a set of devices and a set of tasks achievable with those devices, the model is further refined by a more detailed description of what a 'task' is.

A task is defined inductively as being formed from either an atomic service presented by one of the devices, or any concatenation of atomic services or tasks.

The improvement of a task is then achieved via the substitution of a number of its component tasks (or perhaps the replacement of the task in its entirety) for other tasks, considered to be 'better', according to the improvement criteria.

Obviously, this definition is still quite simple; it does not cater for control flow between component tasks other than a simple sequential approach. Further future work would be in considering how such substitutions could be performed in more complex compositions built with looping and conditional constructs.

The specific problems that must be addressed for this approach to work are related to the description of the tasks and atomic services, such that they may be composed and substituted automatically. There is already significant research, addressing the problem of describing and composing services, in the area of semantic web services.

It is proposed that utilising the semantic web services ontology set "OWL-S" would enable the description, and automatic composition and substitution, of the tasks forming the functionality of an ESE. OWL-S also provides a mechanism for relating the higher abstract descriptions of tasks to the specific functions of devices. It does this in two ways: Firstly, It provides a process modeling ontology that can be used to map from task compositions down to individual services. Secondly, OWL-S includes a grounding ontology that allows for the automatic invocation of the device services from the process model.

In order to demonstrate that this approach is valid an experiment is proposed below, based on the problem of building an ubiquitous resource locator system that improves its functionality as new services are introduced via the addition of devices.

4.1.1 Ubiquitous Resource Locator

An experiment to produce a prototype evolving ubiquitous resource locator using the motivation of constructing a distributed document recommender.

4.1.1.1 Aims

This experiment is intended to show how it might be possible to create an ad-hoc pervasive infrastructure that automatically evolves new or improved functionality as more devices are added.

In the process of implementing a prototype, it is hoped that current shortfalls in the technologies available will be highlighted, thus providing directions for possible further research.

4.1.1.2 Design and evaluation

It was decided that the scope of the experiment should be limited to a specific application as a vehicle for development of the prototype. This restriction of the application area allows for quicker development of the prototype and facilitates the evaluation of the experiment.

The application chosen was a document recommender system. Such a system should recommend documents to a user based on their preferences and context. Choosing this application allows us to specify quantifiable measures of the system's performance that may then be used to evaluate the experiment and indicate how much the systems functionality has evolved.

It should be noted that this experiment is not intended to explore the problem area of document recommender systems and as such, it should not be considered within this research area.

The evaluation of the experiment will be based solely on quantifiable metrics as opposed to more qualitative user tests which fall within the field of HCI, an area that is beyond the scope of this experiment.

Evaluation should consider two aspects of the experiment: firstly, how the overall functionality of the system evolves and secondly, how a system formed from the ad-hoc collaboration of multiple devices performs with respect to standard metrics such as speed and robustness.

In order to evaluate the document recommender functionality it will first be necessary to define the relevant metrics. These will cover aspects such as the suitability of the recommended documents and the total number of documents searched.

Evaluating the systems performance will be done using standard approaches such as measuring the time taken to execute searches for documents and evaluating robustness by examining the systems behaviour as devices are added and removed.

The prototype system will be based around a number of devices, such as a smart phone or PDA, a laptop and a desktop computer. No assumption is made as to the presence (or interconnectivity) of devices. Each device has access to different resources. The desktop computer might have access to a large internet-based document store (perhaps via a search service such as Google) but may not have access to context information for the user. The PDA has more information about the users context (via a GPS receiver and the users diary) but does not have any documents stored locally. The desktop machine may be a shared computer such that the user cannot use it as an interface device however the users laptop is a suitable interface device which can be utilized by the combined system. The document recommender system should be able to make the best use of all the resources available across the devices present.

In order to demonstrate evolving functionality the system should opportunistically utilize the available resources (in the form of the computing devices that are present) in order to provide increasingly improved performance or perhaps even new functionality. In the case of a document recommender system this evolution can be shown in three main areas: the quantity and quality of available documents, the relevance of recommended documents to the user and their context, the display of information to the user (both the document lists and the documents themselves). Other possible avenues for demonstrating new functionality could include offering services such as sharing document lists with other users who may be present, or automatically downloading and printing chosen documents if a printer is discovered. The following scenario illustrates how the system might behave.

Bill is traveling to work by train. With him he has his PDA (on which is stored his diary) and his laptop. In order to prepare for an important meeting that day he is researching related issues by reading some documents he has already downloaded to his laptop. The document recommender system components are running on both the PDA and the laptop, the system understands that Bill is currently reading documents and matches the subject of those documents with the meeting subject in his diary. In order to support his task, Bill is automatically presented with a list of documents on his laptop that may be relevant to the meeting, this list includes some of his own work that he had forgotten and also highlights some slides in a presentation that he could re-use. When Bill arrives at his office his desktop, which is also running some document recommender services, is recognized by his laptop and his PDA. The devices coordinate and the desktop runs further searches for relevant literature on the company's documentation server. The desktop also has access to the shared company diaries and is thereby able to add information about other participants in the meeting to the context model being used to search for documents. It discovers a document written by one of the other participants in the meeting that is relevant. The fact that the document is both relevant and written by a participant gives it a very high relevance score. However, Bill is having a coffee before the meeting and is not currently using his laptop or his desktop. The system therefore makes use of the notification functionality of his PDA to inform Bill of the new document, presenting the title and a short synopsis. Bill sees the notification and decides that the document would indeed by useful to see before the meeting so he returns to his desk where the document is shown on the desktop machine's screen. He decides that he'd rather have it with him in printed form but he does not have a printer so he takes his laptop to the next office where it automatically discovers the printer there and offers him the option of printing the document (which it has automatically downloaded from the desktop computer). Bill has a chance to glance through the document on his way to the meeting and as a result is better informed about the position of one of the other participants - he adjusts his strategy accordingly and successfully negotiates his position at the meeting.

In the above scenario the system demonstrates evolution both through improving functionality (the new resources of the desktop computer being used to extend and better inform the search for documents) and via new functionality (allowing Bill to print the document via the printer in the meeting room).

4.1.1.3 System implementation

The devices involved in the experiment are as follows:

1. Sony Ericsson P900 Smart phone
2. Apple 15" PowerBook Laptop Computer
3. Apple G5 PowerMac Desktop Computer
4. Unspecified Windows-based Computer to act as Bluetooth Lan bridge

These devices will interconnect across three different networking technologies. The smart phone will communicate via Bluetooth, the laptop via 802.11b wireless-LAN and the desktop will be connected to the Internet via the university wired Ethernet network whilst also acting as a bridge to the wireless-LAN network. The Windows-based computer will provide Bluetooth connectivity services to the smart phone, allowing it to communicate with the system via the wired Ethernet network.

Figure 4.1 shows the intended design of the system. The four main devices are shown together with the network connectivity arcs between each one. The three user devices; the smart phone, the desktop and the laptop each have a user model (this may be user preference information or context information). The laptop and the desktop may have documents stored locally and the desktop may also have access to external document stores, perhaps on the Internet. The document recommender system as a whole behaves as if it had access to one combined document store and one combined set of user information.

A timetable of intended future work follows:

Date	Target
December	Implement underlying network layer
January	Continue implementation of network layer Construct domain-specific ontologies
February	Construct set of task descriptions in OWL-S Implement task execution engine
March	Implement task substitution system
April	Gather experimental results
May	First draft of mini-thesis
June	Continue work on mini-thesis
July	Submit mini-thesis

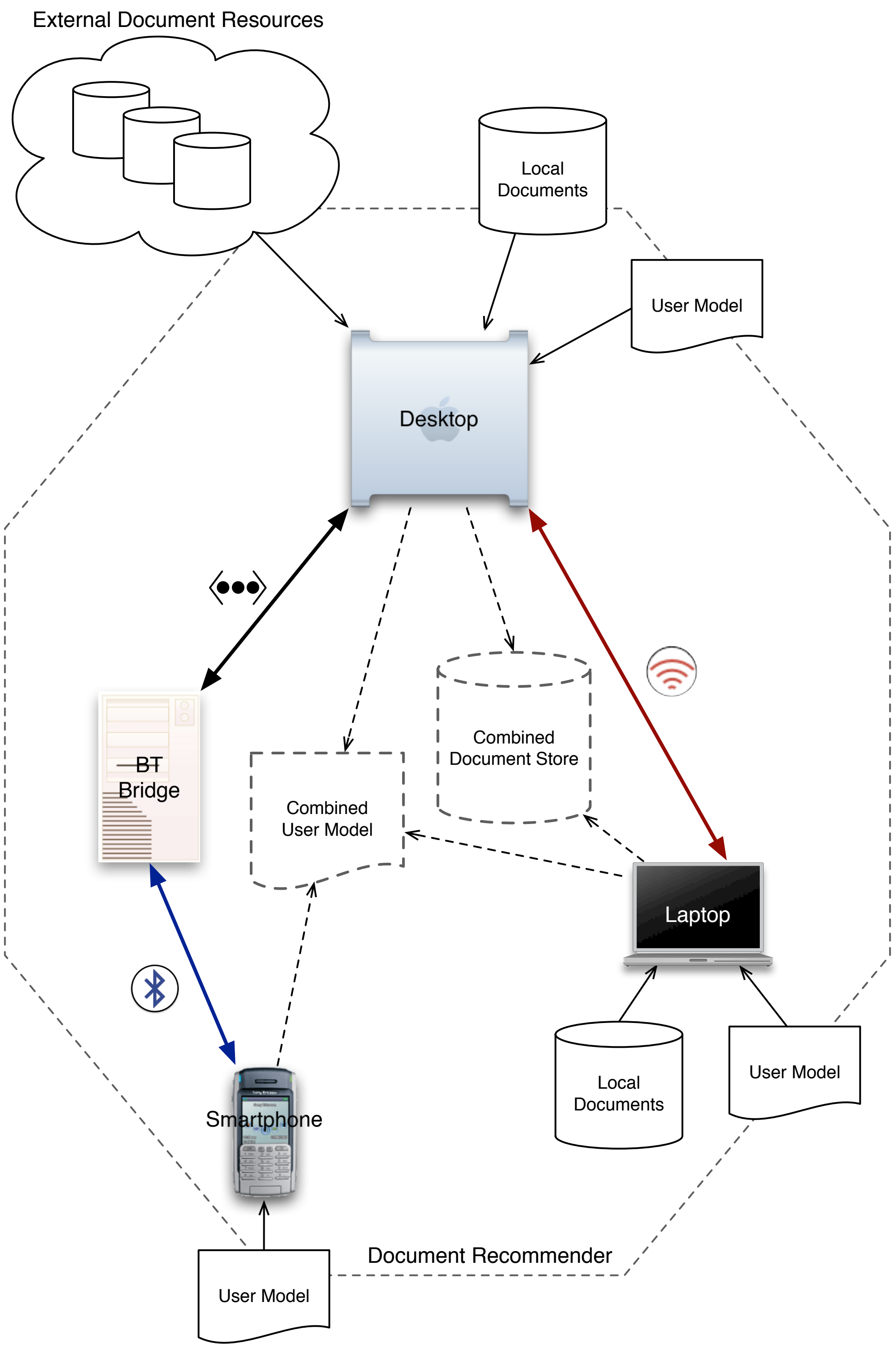

FIGURE 4.1: Experiment Design

Bibliography

Ankolekar A, Burstein M, Hobbs J R, Lassila O, Martin D, McDermott D, McIlraith S A, Narayanan S, Paolucci M, Payne T & Sycara K (2002). DAML-S: Web Service Description for the Semantic Web, *in* I Horrocks & J Hendler, eds, *The Semantic Web - ISWC 2002: First International Semantic Web Conference*, Springer-Verlag, p. 348.

Berners-Lee T, Hendler J & Lassila O (2001). The Semantic Web, *Scientific American* .

Chen H, Finin T & Joshi A (2003). An Ontology for Context-Aware Pervasive Computing Environments, *Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review* .

Chen H, Tolia S, Sayers C, Finin T & Joshi A (2001). Creating Context-Aware Software Agents, *First GSFC/JPL Workshop on Radical Agent Concepts* .

Coen M H (1998). Design Principles for Intelligent Environments, *in Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98).*

Dale J & Mamdani E (2001). Open Standards for Interoperating Agent-Based Systems, *Software Focus* **2**(1), 1–8.

Gandon F L & Sadeh N M (2004). Semantic Web Technologies to Reconcile Privacy and Context Awareness, *Web Semantics Journal* **1**(3).

Heflin J & Hendler J (2001). A Portrait of the Semantic Web in Action, *IEEE Intelligent Systems* **16**(2), 54–59.

Hendler J (2001). Agents and the Semantic Web, *IEEE Intelligent Systems* **16**(2), 30–37.

IETF (2004), 'Extensible Messaging and Presence Protocol (xmpp) Charter'.
URL: *http://www.ietf.org/html.charters/xmpp-charter.html*

JabberD (2004), 'jabberd project'.
URL: *http://jabberd.jabberstudio.org/*

JCP (2004), 'The Java Community Process(SM) Program - JSRs: Java Specification Requests - detail JSR 82'.
URL: *http://www.jcp.org/en/jsr/detail?id=82*

Jennings N R (2000). On agent-based software engineering, *Artificial Intelligence* **117**(2), 277–296.

Jennings N R (2001). An agent-based approach for building complex software systems, *Communications of the ACM* **44**(4), 35–41.

JSF (2004), 'Jabber: Open Instant Messaging and a Whole Lot More, Powered by XMPP'.
URL: *http://www.jabber.org/*

Kahn J M, Katz R H & Pister K S J (2000). Emerging Challenges: Mobile Networking for "Smart Dust", *Journal of Communications and Networks* **2**(3), 188–196.

Lassila O, van Harmelen F, Horrocks I, Hendler J & McGuinness D L (2001). The semantic Web and its languages, *IEEE Intelligent Systems* **15**(6), 67–73.

Martin D, Paolucci M, McIlraith S, Burstein M, McDermott D, McGuinnes D, Parsia B, Payne T, Sabou M, Solaki M, Srinivasan N & Sycara K (2004). Bringing Semantics to Web Services: The OWL-S Approach, *in Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, San Diego, California, USA.

McGuinness D L, Fikes R, Hendler J & Stein L A (2002). DAML+OIL: An Ontology Language for the Semantic Web, *IEEE Intelligent Systems* **17**(5), 72–80.

Moreau L (2002). Agents for the Grid: a Comparison with Web Services (Part I: Transport Layer), *in* K L H. E. Bal & A Reinefeld, eds, *Second IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID 2002)*, IEEE Computer Society, IEEE Computer Society, pp. 220–228.

Pinsdorf U, Peters J, Hoffmann M & Gupta P (2002). Context-Aware Services based on Secure Mobile Agents, *in* N Rozic & D Begusic, eds, *10th International Conference on Software, Telecommunications & Computer Networks (SoftCOM 2002)*, IEEE Communication Society, Ministry of Science and

Technology Republic of Croatia and University of Split, University of Split, R. Boskovica, HR-21000 Split, Croatia, pp. 366–370.

Psi (2004), 'Psi Jabber Client:: home'.
URL: *http://psi.affinix.com/*

Ramchurn S D, Deitch B, Thompson M K, Roure D C D, Jennings N R & Luck M (2004). Minimising Intrusiveness in Pervasive Computing Environments using Multi-Agent Negotiation, *in Proceedings of the First Annual International Conference on Mobile and Ubiquitous Systems*, Boston, Massachusetts.

Román M, Hess C, Cerqueira R, Ranganathan A, Campbell R H & Nahrstedt K (2002). A Middleware Infrastructure for Active Spaces, *IEEE Pervasive Computing* .

Sadeh N M, Chan E & Van L (2002). MyCampus: An Agent-Based Environment for Context-Aware Mobile Services, *in AAMAS - First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, ACM Press, Bologna, Italy.

Sashima A & Kurumatani K (2002). Seamless Context-Aware Information Assists Based on Multiagent Cooperation, *in Proc. of The second International Workshop on Agent-based Approaches in Economic and Social Complex Systems*, pp. 39–46.

W3C (2004*a*), 'OWL Web Ontology Language Overview'.
URL: *http://www.w3.org/TR/owl-features/*

W3C (2004*b*), 'RDF Primer'.
URL: *http://www.w3.org/TR/rdf-primer/*

Weiser M (1993). Some Computer Science Issues in Ubiquitous Computing, *Communications of the ACM* **36**(7), 74–84.

Wooldridge M (2002). *An Introduction to MultiAgent Systems*, John Wiley & Sons, England.