massachusetts institute of technology — artificial intelligence laboratory

Delegation, Arbitration and High-Level Service Discovery as Key Elements of a Software Infrastructure for Pervasive Computing

Krzysztof Gajos and Howard Shrobe

AI Memo 2003-016 June 2003

Abstract

The dream of pervasive computing is slowly becoming a reality. A number of projects around the world are constantly contributing ideas and solutions that are bound to change the way we interact with our environments and with one another. An essential component of the future is a software infrastructure that is capable of supporting interactions on scales ranging from a single physical space to intercontinental collaborations. Such infrastructure must help applications adapt to very diverse environments and must protect people's privacy and respect their personal preferences. In this paper we indicate a number of limitations present in the software infrastructures proposed so far (including our previous work). We then describe the framework for building an infrastructure that satisfies the abovementioned criteria. This framework hinges on the concepts of *delegation*, *arbitration* and *high-level service discovery*. Components of our own implementation of such an infrastructure are presented.

This work was supported by Acer Inc., Delta Electronics Inc., HP Corp., NTT Inc., Nokia Research Center, and Philips Research under the MIT Project Oxygen partnership and by DARPA through the Office of Naval Research under contract number N66001-99-2-891702.

1 Introduction

Numerous systems have been proposed as software middleware for pervasive computing (see Section 5 for a brief survey). Many of these system provide mechanisms for resource discovery and some of them claim that these mechanisms are intended to scale to extremely wide area pervasive computing environments. In practice, most of these system, including our own previous work, have actually dealt with simple scenarios involving, for example, a few users in a single physical space or a small number of people with mobile personal computing devices. These limited scenarios fail to raise a set of issues that become critical when the set of users and spaces expands drastically. We believe that very few of the existing systems possess the mechanisms necessary to deal with these contexts.

We begin this paper by presenting a simple scenario similar to those previously used by ourselves and other researchers; we then identify a number of issues raised by this scenario and show that these issues cannot be addressed without new mechanisms that enhance the capabilities of the existing systems. Finally, we present our own answers to these problems in the form of subsystems for high-level service discovery, delegation and arbitration.

1.1 Scenario

Let us consider prevalent approaches to building pervasive computing software infrastructures by analyzing a simple scenario, very similar to those presented Mark Weiser [28], MIT Project Oxygen [9] or the Portolano Project at University of Washington [1].

> Anne is sitting in one of the shared work rooms in her office building. She is alone, preparing a presentation. Her work is displayed on one of the two large displays in the room. She also has her handheld device with her. Meanwhile Bob is sitting in his office reading news. At some point he comes across an item he wants to share with Anne. He asks his office to connect him with Anne. The room Anne is working in tells her she has a communication request from Bob, and asks if it should make the connection. Anne agrees to a video connection and Bob's face shows up on the unoccupied display. After a brief exchange, Bob asks his office to forward the interesting news item to Anne and it gets displayed on her handheld.

1.2 Problems

We would like to concentrate here on how Bob's office can locate appropriate communication devices in Anne's vicinity regardless of whether she is in her office, a shared work room, or just taking a walk in the park with only her universal handheld device. Some suggest that Bob's system should discover a communication device near Anne and try to establish a connection with it. The infrastructure proposed for the Portolano Project [10] would send an event addressed descriptively to an appropriate device. It would be the responsibility of the network to route the information to the correct destination. A similar solution is proposed by the Intentional Naming System [3] with the added flexibility of choosing between early binding (where Bob's system obtains the address of the device and communicates with it directly) and late binding (where Bob sends information to the descriptive address and every time the appropriate destination is chosen by the network). There are several problems with such approaches:

1. *How can the infrastructure know what the devices in Anne's vicinity are?* Note that in a one-step discovery process proposed by INS, as well as in data-centric network routing proposed

by Portolano, the devices in Anne's vicinity would need to advertise themselves as being close to Anne. Alternatively, Anne would need to advertise her location and then Bob's system would look for devices at that location. Both solutions hinge on reducing Anne's privacy – her location can be deduced from the information available in the system. Both solutions also require updates to the advertised descriptions of either the devices or Anne every time she changes her location. Finally, both solutions require that all devices advertise themselves to the global discovery network.

2. *How can we ensure that Anne always has the right to accept or reject the connection?* If Bob tries to establish communication with devices in Anne's environment and not with Anne herself, how can we ensure that Anne has the ability to accept or reject a call? Or, even more fundamentally, if an hour earlier, while still in her office, she said to her environment, "I do not want to respond to any calls for the next two hours," how can we ensure that the new environment she is in also respects her wish? What is the pervasive computing equivalent of the *off* button on a cell phone?

3. *Who gets to decide how the information is presented to Anne?* Even if we extended the discovery process with a step where Bob's software has to ask Anne explicitly about her location giving her a chance to decline an answer or to just point Bob to her "mobile environment," it is still not clear how Anne can impose her personal preferences on how she wants to communicate with Bob, and how the information he sent her is presented to her. Anne may prefer short text items to be spoken to her rather than displayed. And if they are displayed, she may have a preference regarding which devices are to be used. In a one-step discovery mechanism there is no clear place where such options could be easily defined, especially if we take into account the fact that Anne can be in an anonymous location, such as a shared work room, or in her private office or home (which, in contrast to the work room, can be permanently configured to suit her needs). Note that "loading Anne's preferences" does not necessarily solve the problem unless the devices in a space can change how they advertise themselves to ensure that the right ones will be chosen for the right tasks.

4. *How do we avoid resource conflicts?* In other words, how do we make sure that when the connection between Bob and Anne gets established, the free display is used as opposed to the one taken by Anne's presentation? And when Bob sends over the news item, how do we ensure that it gets displayed on the only remaining free display? In a one step discovery mechanisms it would be possible to change the descriptions of all of the used resources to indicate that they are currently not available. But in such a case, what would happen if Anne did not have her handheld with her? How could the system decide on its own which of the occupied displays to borrow?

5. *What do we do if Anne is in an environment where the access control mechanisms permit her to access all of the devices but deny access to Bob?* When pervasive computing becomes truly pervasive, access control mechanisms will need to be put in place to ensure that a prankster next door does not make my stereo play music in the middle of the night or that telemarketers do not put information about their latest products on all the walls of my office. In such an environment, how can we make it clear that under those particular circumstances presented in the scenario, Bob can access a device for displaying information to Anne?

In short, currently available software infrastructures would have problems scaling up to large numbers of people and physical spaces. Furthermore, current mechanisms largely violate people's privacy, social norms and do not respect individual preferences.

2 The Ideal Infrastructure

We can synthesize observations made in the previous section by listing the most important requirements for a resource management middleware for pervasive computing.

2.1 Adaptivity

There are several aspects of adaptivity that the software infrastructure should address. First, software should be able to adapt to *new environments*. The same application should be able to function in a highly-instrumented conference room, an office, a living room, a car or even on a small portable device that the user is likely to travel with. Different kids of environments are very likely to have completely different kinds of hardware and software resources available. A conference room might have large public displays, smaller displays for individual participants, speech capabilities as well as peripheral information presentation devices such as scrolling LED signs. A car, while being driven, would only have speech available for many applications, displays being deemed too distracting for safe driving. In short, it is critical to perform resource management at the level of services such as information delivery rather than at the level of particular devices such as a display.

Software also needs to adapt to *changes in the environment*. Environments may change dynamically: devices may come and go at any time and some components may fail. Hence the resource management service needs to observe the changes to the set of available services in order to, on one hand, make sure that requesters may get better resources as they become available and, on the other hand, that failing components are replaced on the fly with functioning substitutes.

Finally, the choice of methods for satisfying service requests needs to depend on the current activity context. For example, important email alerts may be delivered by speech if one is reading a book, but visual environmental devices should be used if the person is on a phone, and a personal handheld while in a meeting.

2.2 Scalability

There are two distinct ways in which we wish to address scalability. The first has to do with the sheer *numbers of participating components*. A global pervasive computing system would need to deal with millions of people and probably orders of magnitude more devices. It needs to be possible for a person to interact with all of the devices in his or her immediate surroundings and for people to locate one another across continents.

In addition, we need to address the scalability at the level of the *number of simultaneously executing applications* in any single environment. Large numbers of applications running concurrently are likely to contend for the same resources. In order to cope with the scale of complexity of interactions in pervasive computing, we need to provide arbitration mechanisms that will allow applications running in the same environment to coexist “peacefully” and to make the best use of available resources.

Consider the following example: in an office equipped with an on-wall projector and a TV set, I request to watch the news. The projector is assigned to the job because it produces the largest image and has the best resolution. Then, while watching the news, I request to have access to my email agent. This agent has to use the projector because this is the only display that can be used by a computer. Therefore the projector is taken away from the news. The change may be made less inconvenient, however, if instead of stopping the news, the system moves it to the TV set. We discuss this topic in detail in [13].

2.3 Respect For Privacy And Personal Preferences

Privacy and security are important issues in computing and they cannot be ignored in development of pervasive computing infrastructures. In particular, resource discovery mechanisms need to work closely with access control tools. Whenever a request is made for a service or a resource, access control mechanisms need to evaluate what resources, if any, the requester is allowed to have access to given the identity of the requester, the current state of the resource, the location of the requester and many other conditions [26]. At times, the user will have to be asked to make a decision [4] but in any case, the user needs to be fully informed how his or her personal information (such as location, current activity) is being used [2].

As argued before, the software needs to adapt its actions to the different environments it might be running in. It also needs to adapt itself to the personal preferences of individual users. For example, I may prefer not to receive any communication requests while in a meeting with the exception of urgent requests from my family. Such requests, if they happen, should be satisfied with a text-based interface rendered on my handheld device. It is important to note that such preferences would be very difficult to enforce in a world, where potential callers communicate directly with the devices in my surroundings.

It is important thus, to structure the middleware in such a way that the access control and personal preference mechanisms can influence the handling of service requests and the allocation of resources.

2.4 Ease of engineering

As argued by Pell, et al. [22] and by Gajos, et al. [13], a high-level resource management abstraction makes it easier to build large systems out of individual components. Many of the interactions do not need to be defined by the designer but, instead, are determined at runtime by the system itself.

3 Building The Ideal Infrastructure

We have used a new approach to building pervasive computing middleware that addresses the issues discussed in the previous section. The infrastructure we have already mostly deployed is designed to provide the following three mechanisms:

Delegation In short, let Anne make decisions that concern Anne. Thus we need to establish an abstraction barrier that coincides with the boundaries of what concerns any entity (be it a person, a physical space, an institution or a group of people). Entities have the autonomy of deciding if and how they wish to interact with the rest of the world.

High-level service discovery Discovery and negotiation are performed at a high enough level of abstraction to allow adaptivity and flexibility in exercising personal preferences.

Arbitration Arbitration mechanisms are established within each entity to allow numerous tasks to coexist.

In Section 4 we outline the engineering details of the components of this infrastructure. First, however, we discuss the abovementioned mechanisms individually.

3.1 Delegation

For convenience, we introduce the concept of a "society" [23, 8] to mean a collection of software components acting on behalf of a single entity. Hence, all of the software acting on behalf of Bob, such as his scheduler, information delivery and management, email alert system, etc, are understood to belong to a single society. The software managing interactions in his living room belongs to a different society. There can also be societies representing information providers, institutions, groups of people, etc.

In our system, we separate the high-level resource discovery and management processes taking place within and among societies. The basic principle is simple: if we want to find a service that does scheduling for Bob, we first need to locate Bob's resource discovery and management system. It is then the responsibility of Bob's software to decide how (and if) the scheduling service is to be provided. If we need to discuss a product purchase, we first contact the manufacturer and request a sales service. It is then up to the manufacturer to decide how to fulfill the request: we may be directed to one of the sales specialists or to an automated sales agent.

The advantages of such arrangement are three-fold. First of all, it allows the society's owner to make the decisions about the access to its/his/her resources and information. If the society represents a physical space, it can then use its access control policies to decide what service requests to grant and which to deny. It can also decide whether or not it wants to divulge the information about what resources it has at its disposal: after all it would not be desirable to advertise to the entire world all of the A/V components and displays one has in one's living room. Yet if a friend calls, we do want to use that high-definition digital TV. If a society is owned by a person, the resource management system of that person can decide if to respond to service requests and if so – how. In both cases, as argued in [4], the decision may be referred all the way up to the person if the system determines it does not have the necessary information to decide automatically.

Finally, hiding all the individual devices and services from the global (world-wide) discovery system, greatly reduces the scale at which the global discovery needs to be performed. Firstly, the sheer number of elements registered with the global discovery service is reduced. Secondly, the complexity of the descriptions can be reduced: instead of advertising all the parameters of every device, it may be sufficient to advertise the name and kind of each entity with the option of naming some of the major services provided (for example, shared conference rooms may want to advertise their size, and kinds of capabilities).

Also, the concept of meta-societies representing institutions and groups of people, allows us to hide from the rest of the world many of the internal resources available within such meta-societies.

In case of our Lab, for example, only the public resources, such as the publications service, and the identities of individual researchers should be visible to the world. Other resources, such as conference rooms, A/V equipment, or location of the espresso machines are kept internal to the Lab.

3.2 High-level service discovery

Many of the systems summarized in Section 5 have the capability for describing resources at a high level as the semantics of the description language is not restricted; few of them, however, suggested using descriptions at a level higher than a single taxonomy: printer, display, audio input. As argued before, communication in terms of high level tasks or services is necessary for applications to be able to adapt to a wide variety of environments and to individual user's preferences. If a request is for a display device, we can only choose between a projected display or a handheld device. If, on the other hand, the request is framed in terms of delivering a short text message, it can be rendered through speech, display on an LED sign or on a private device, or even by printing it on a printer. The choice of the method would depend on the available resources and on the preferences of the recipient of the message.

We are not unique in suggesting that device-level descriptions are insufficient in pervasive computing. For example, Winograd [29] argues against using the concept of device drivers, Schubiger-Banz et al. [25] suggest "addressing by concept" in all ubiquitous computing environments (both spaces and/or collections of mobile devices) and the aforementioned INS [3] uses "intentional names" for all networked resources.

3.3 Arbitration

Arbitration is essential in any larger system that allows multiple applications running simultaneously to share scarce resources. It allows individual applications to be written without having to take other applications' needs into consideration. Most of the current pervasive computing projects demonstrate the capabilities of their systems using single task scenarios, which allow researchers to ignore the problem of resource conflicts altogether. As pervasive computing becomes a fact of everyday life, independently written applications such as teleconferencing, entertainment, news updates, etc, will have to function simultaneously in the same environments. The middleware needs to ensure that applications do not "steal" resources from one another unnecessarily.

3.4 Back To The Scenario

Let us return to the scenario from Section 1.1 and illustrate how that interaction could happen if we incorporated the mechanisms proposed above.

Figure 3.4 illustrates the discovery steps taking place in this scenario. In our system, when Bob asked his system to contact Anne, his communication agent would first reserve the necessary communication resources in Bob's environment (steps 1 and 2). It would then request of Bob's resource manager a service for communicating with Anne (step 3). Bob's resource manager would use the global discovery system to establish communication with Anne's resource manager and it would then make a request for a communication service of Anne's resource manager (step 4). Anne's software would then verify that Bob was allowed to make such a request. Assuming that the answer was positive, it would return a handle to an appropriate service. Bob's software would communicate with Anne's communication agent which, in turn would make a request of its own resource manager for audio and video devices (step 5). Anne's resource manager would realize that it controls only a low quality screen and a camera on the handheld device, but knowing that Anne was in the work room, it would ask the work room for better resources (step 6). The room, having one display free, would allow Anne's software to use it. Later, when Bob tried to show a piece of news to Anne, Anne's resource manager would again check with the work room if it had any available resources. Given that both displays in the room were taken, Anne's resource manager would decide that the difference in quality between Anne's handheld device and a large display was not crucial in this case and would resort to the small screen on the handheld.

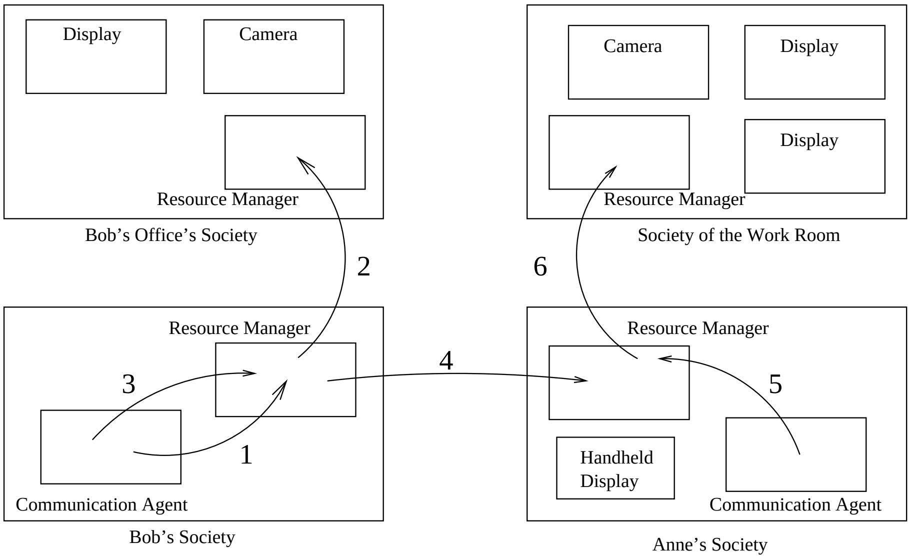

Figure 1: Conceptual flow of the discovery process: initially Bob's communication agent asks Bob's resource manager for appropriate resources needed to establish a video call to Anne (step 1). Bob's resource manager forwards the request to the resource manager of Bob's office (step 2). Once these resources are reserved, Bob's communication agent gets in touch with Anne's communication agent – the process is mediated by Bob's and Anne's resource managers (steps 3 and 4). Finally, Anne's communication agent obtains appropriate resources from Anne's surroundings (steps 5 and 6).

It is assumed that Anne's software does not really need to be collocated with her. In fact, parts of her society are probably running at a static location at her home or office thus aiding dicovery.

The multiple steps of delegation arguably make the protocol more complex but let us revisit the problems raised after this example was originally presented:

1. *How can the infrastructure know what the devices in Anne's vicinity are?* This is no longer necessary – all that Bob's software needs to do, is discover the entry gateway into Anne's "society." Once this has been accomplished, it is the responsibility of Anne's software to find out where Anne is and what devices are available to her.

2. *How can we ensure that Anne always has the right to accept or reject the connection?* In the framework proposed here, the request for a connection service comes to Anne's software and not to the devices surrounding her. Hence it is up to Anne to set up her software in a way that it makes automated decisions when appropriate and asks her otherwise. If she has instructed her software an hour earlier that she did not wish to take any calls for the next two hours, her software would automatically deny Bob's request, and there would be no need to transfer Anne's preferences from one location to another.

3. *Who gets to decide how the information is presented to Anne?* Now the decision is clearly in Anne's hands. Anne's resource manager decides whether to use any of Anne's personal

devices or to forward the request to whatever environment Anne is in. The request for a conversation from Bob was delivered to Anne's communication agent. That agent, in turn, could decide how to handle the request: whether to use just audio or both audio and video. In either case it also got to decide whether to accept the connection at all. In case of the news item provided by Bob, Anne's personal preferences might cause the system to try to display the item (as happened in the scenario) but if she were driving, the system would be more likely to offer to read the news item to her instead.

4. *How do we avoid resource conflicts?* Organizing resources into societies makes it possible to build resource management mechanisms that oversee all resources within each society in a conceptually centralized manner. This arrangement makes it possible for Anne's resource manager to see all of the devices available to it (i.e. the handheld device) and the best resource offered by the work room and to decide what to use.

5. *What do we do if Anne is in an environment where the access control mechanisms permit her to access all of the devices but deny access to Bob?* When information is delivered to Anne's information presentation service, the subsequent requests for devices come from Anne's software and not from Bob's. Once Anne decided to accept the communication request, it is now up to her to obtain appropriate resources. The resources are requested on Anne's behalf and not Bob's.

4 Engineering Details Of A Scalable Pervasive Computing System

For several years the Intelligent Room Project [16] has been developing components of a pervasive computing infrastructure that fit within the framework described above. We have extensive infrastructure to support communication, discovery and arbitration within individual societies. We have also implemented a prototype system for inter-society discovery and communication, and we are currently working on a negotiation mechanism that will allow resource managers of different societies to negotiate use of scarce resources. In the following sections we will describe the individual components of our infrastructure in more detail.

4.1 Metaglue

Metaglue [23, 8] is a multi agent system implemented in Java. It provides the low level communication infrastructure for each society. It has a low-level discovery service, called the “Catalog”, that allows agents to locate one another on the network based on agents' unique identifiers. Metaglue supports direct communication via remote method calls using Java RMI mechanism. The coupling among agents is not as tight as the mechanism might imply – the method calls are mediated through special proxy objects [27], which, in case of the failure of the destination agent, can contact the Catalog to retrieve the new location of the failed agent or the closest substitute. The proxy objects are also used by the resource management system (described below) to occasionally swap the resources granted to agents. Hence, although agents keep executing method calls on the same proxy object, the real targets of the calls may change. The agents are, of course, notified of significant changes.

In addition to synchronous method calls, Metaglue has mechanisms for a publish-subscribe model of communication. It also provides mechanisms such as persistent storage, automatic restarting of failed components, customization and support for multimodal input and output.

Metaglue has been successfully deployed in a number of locations: a small conference room, a small living room, a number of student and faculty offices as well as a bedroom and an information kiosk [17]. It was the deployment of our software in a number of different locations that made the need for more adaptive mechanisms real. It also motivated the work on wide-scale discovery mechanisms that would allow various spaces and users to interact with one another.

4.2 Rascal

Rascal [12] performs two major functions within a Metaglue society: high-level resource discovery and arbitration among requests for services. It is composed of three major parts: the knowledge base (implemented in Jess [11]), the constraint satisfaction engine (JSolver [7]), and the framework for interacting with other Metaglue agents. Resource discovery is performed entirely by the knowledge-based component of Rascal. Arbitration begins in the knowledge-based part (where relative cost and utility of various resources are determined) but most of the work on arbitration is done by the constraint satisfaction engine. The components for interacting with the rest of the Metaglue agents facilitate communication with service providers and requesters, and enable enforcement of Rascal's decisions (i.e., taking previously allocated services away from requesters).

Upon startup, information about all available resources is loaded into Rascal's knowledge base (if more resources become available later on, they are added dynamically). Rascal relies on all resources having the descriptions of their needs and capabilities separate from the actual code because components in Metaglue are started on demand so Rascal needs to be able to reason about their capabilities even before they are brought to life. Those external descriptions provide a list of services that the agent or other resource can provide. For each service provided, agents may in addition specify what other resources they, in turn, will need in order to provide the service. For example, the `MessengerAgent` that provides a message delivery service will need one or more resources capable of providing text output service. Agents may also specify their startup needs, i.e. a list of requests that need to be fulfilled for the agent to exist. For example an agent providing speech recognition service will need a computer, with appropriate speech recognition software installed, in order to be able to start and configure itself properly.

When Rascal considers candidates for a request, it not only needs to make sure that those candidates are adequate and available – it also needs to make sure that the needs of those candidates can be satisfied, and that the needs of the resources satisfying the needs of the candidates can be satisfied as well, and so on. The final selection of candidates for requests is performed by the constraint satisfaction engine. Therefore the knowledge-based part needs to evaluate all possible candidates for all possible requests. This request chaining proves to be extremely valuable: when the email alert agent, for example, requests a text output service, several different agents are considered: for example the LED sign and the speech output. The email alert agent may have its own preference as to what kind of rendition of the text output service it prefers. However if the communication link with the actual LED sign is broken, the needs of the agent controlling the LED sign will not be satisfied and so it will not be assigned to the request.

Arbitration in Rascal is based on need estimates provided with each service request, in conjunction with the concept of utility of a service to the requester and the cost to others. This is a very simple and arbitrary scheme. It could easily be replaced by a different system (e.g. market-based

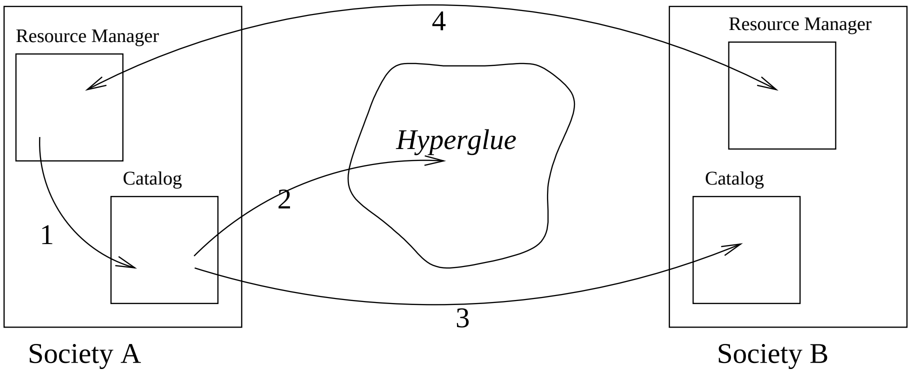

Figure 2: The most common use of Hyperglue. If a resource manager of society A needs to negotiate resource use with society B, it requests of its local Catalog a handler for communicating with the resource manager of B (step 1). The Metaglue identifiers for resource managers follow a standard format so this process can be referred to the low level lookup service such as the Catalog. The Catalog of society A then realizes that it does not know how to communicate with B so it queries Hyperglue for the network location of B's Catalog (step 2). After this query is fulfilled, the Catalog of A contacts its counterpart in B (step 3) and obtains the handler for B's resource manager. After that the resource managers of the two societies can communicate directly (step 4).

mechanisms such as [5]) should there be a need for that. This simple model is sufficient for the current implementation of Rascal if we assume that all agents within a society can be trusted.

The basic assumption of this schema is that, given a request, each candidate resource has some utility to the requester. This utility depends on how badly the requester needs a particular request fulfilled and on how well the resource matches the request. The same method is used to calculate the utility of the already allocated resources to their owners. When a resource is taken from its current user, the system as a whole incurs a cost equal to the utility of that resource to its ex-user. Also when a resource, currently allocated to fulfill a request, is replaced with a different resource, cost is incurred. This cost is a sum of a fixed "change penalty" and the difference in utilities between the new resource and the old one (if this difference is negative, it is set to zero).

The arbiter has to make sure that whenever it awards a resource to a new request, the cost of doing so should never exceed the utility of the awarded resources to the new requester. Rascal provides a number of methods for calculating utility and for evaluating matches between requests and resources. Each resource or request description can also be accompanied by its own custom tools for performing those calculations.

4.3 Hyperglue

Hyperglue has been designed as a low-level inter-society discovery mechanism. It uses the Intentional Naming System (INS) [3] at its core. As argued before, INS is not sufficient as *the* resource discovery mechanism for pervasive computing but it can be successfully used as a dynamic, wide-scale system for locating entry points of individual societies.

Figure 2 shows the most common use of Hyperglue. If a resource manager of society A needs to negotiate resource use with society B, it requests of its local Catalog a handler for communicating with the resource manager of B (step 1). The Metaglue identifiers for resource managers follow a standard format so this process can be referred to the low level lookup service such as the Catalog. The Catalog of society A then realizes that it does not know how to communicate with B so it queries Hyperglue for the network location of B's Catalog (step 2). After this query is fulfilled, the Catalog of A contacts its counterpart in B (step 3) and obtains the handler for B's resource manager. After that the resource managers of the two societies can communicate directly (step 4).

This picture illustrates the distinction between the low-level and the high-level discovery mechanisms. The former is used to establish the communication between societies and their resource managers, while the latter is used to negotiate if and how user tasks should be performed.

4.4 Inter-Society Service Requests

We are currently designing the protocol for resource managers of individual societies to negotiate resource use among themselves. This protocol is an extension of the one used to request resources within a society. The additions have to do with the fact that when two resource managers communicate, neither of them knows about all the resources that are available to the two of them. If we look back at the scenario, when Bob sent Anne a news item to look at, Anne's resource manager knew that it had a little handheld device that was not being used. It knew also that it was only a good but not perfect resource for the task. It asked the resource manager of the work room then to find out if it could offer it a better alternative. The two resource managers needed to communicate to one another that Anne's society had a free resource of medium quality, that the resource request was of average importance, and that the work room had better resources available but they were currently in use. If Anne did not have her handheld and if it was really important for her to see the news item, Anne's resource manager could have negotiated the use of one of the displays given that Anne's presentation did not need a display while Anne was talking to Bob.

The situation gets even more complicated when there is more than one person in a room and each needs resources.

5 Survey Of Existing Infrastructures

The Open Agent Architecture (OAA) [21] is one of the oldest middleware systems that was suggested explicitly for developing interactions in smart spaces. As the name implies, OAA is an agent based system, where agents, providing various capabilities are organized into larger applications at runtime. All communication in OAA is conducted through a central agent called *the facilitator* which registers agents' capabilities and brokers requests. Requests are expressed as tasks that can be arbitrarily complex. The facilitator will often break them down into simpler sub-tasks before delegating them to appropriate agents. OAA lacks explicit mechanisms for reserving resources and for arbitrating among conflicting requests. This seems to stem from the fact that OAA is primarily used for building complex multi-modal applications (e.g. [6]) but rarely for constructing systems involving several independent applications. It is implicitly assumed in OAA that all agents own the physical resources they use and that all tasks are atomic and can be serialized. OAA is apparently not intended to scale beyond a single environment.

The infrastructure behind the SmartOffice project [15, 14] is organized similarly to OAA: distributed components communicate through a centralized *supervisor*. The communication is somewhat more loosely defined than in OAA: messages can be announcements of events (recognition event, system event) or requests for tasks or resources. The communication is defined at a high level and allows loose coupling of components. Somewhat surprisingly, the middleware of SmartOffice does not address the issue of arbitration even though it is supposed to mediate all interactions happening in a smart space. The examples presented in the papers include many tasks that are nearly point-like in time (such as requesting a camera to obtain a picture of an entering person) and no mention is made of how the infrastructure would cope with activities that require long time ownership of resources (such as watching a movie or listening to an email being read to the user).

Smart Platform [31] is an architecture currently used in the Smart Classroom project [30]. The project initially used OAA but recently proposed a new approach. Smart Platform, uses a "hybrid" communication scheme: messages are sent through a central broker using a publish-subscribe mechanism, but streamed data is sent directly between components. Current descriptions of Smart Platform do not address the issues of resource or task management directly. Only lower level communication protocols are described. The project proposes to use two kinds of speech acts for all messages: the *inform* and *query*, thus suggesting a high-level communication model, which would allow loose coupling.

The Gaia Operating System [24] has a discovery and resource management component [18] that deals with resources at the level of CPU cycles, available memory and communication bandwidth. It employs a number of mechanisms that are of interest to us. Most importantly, unlike many other discovery systems, it allows the components to specify in their descriptions not only services they provide but also the services they, in turn, need to function properly. They are called the "prerequisites." This allows the resource manager to allocate only those resources to a request that are likely to be really able to provide the service. Unfortunately, it is unclear how the system deals with multiple applications contending for the same resources. On one hand there is a mention of "reservations" but on the other hand there is a description of a mechanism for notifying applications if another application has taken over some of the resources previously allocated to them (thus implying that the reservations are not necessarily honored).

One of the solutions proposed for the Portolano project [1] involves the concept of "data-centric networking" [10]. In such a system, various communicating components would no longer send data packets to individual network addresses. Instead, they would send events addressed by complex descriptions of capabilities of the receiver. The network would actively route the events toward appropriate destination.

The Intentional Naming System (INS) [3] is a resource discovery system, in which various participating resources advertise their capabilities as opposed to their network address. INS was intended as a more dynamic and more expressive alternative to DNS. Typically, a resource such as a printer, would advertise its location, basic capabilities as well as current queue length to the INS. INS can then use this information to route queries such as "find me a least-loaded color printer nearby." INS supports two options for discovery: early and late binding. The first returns an network address of a resource in response of a query. The latter finds the best match for a description every time a message is sent to it. Late binding offers added robustness in cases when a device changes its location or if a better device becomes available. This system is, at least in principle, supposed to scale beyond a single organization. According to the current design, all participating devices are visible throughout the network.

The SHARP Resource Coordination Protocol takes a very different view of resources and resource management in an agent system controlling an intelligent home [20]. SHARP is the only system described in this paper that operates in a simulated rather than real environment [19]. As a consequence, its designers could ignore some of the real-world challenges that other groups needed to address but could, instead, attempt to identify further challenges. In contrast to other systems described here, SHARP manages such resources as hot water, noise, coffee beans and other physical products and needs of agents operating in an intelligent home. In terms of mechanisms, SHARP assumes that in case of resources such as hot water, the agent responsible for heating the water would be in charge of managing it. Agents are allowed to make reservations but the concept of priority allows agents with urgent needs for a resource to override reservations made by other agents. Other resources, for example noise, are supposed to be managed by all concerned agents collectively. The exact mechanism was not specified. One striking limitation of SHARP has to do with a lack of mechanism for managing a resource if more than one agent can provide it. Hence if we added a second water heating agent in the house, it is not clear how the two agents would negotiate which one of them should fulfill which requests to ensure maximum utility of the entire system.

6 Contributions

In this paper we have indicated a number of ways in which prevailing approaches to building software infrastructure for pervasive computing fall short of delivering on their promise. In particular we have indicated that low-level resource discovery mechanisms proposed by a number of projects are not sufficiently adaptive, scalable or respectful of social norms, personal preferences, or protective of people's privacy.

We have sketched out a framework for building an infrastructure that, we believe, will allow us to make pervasive computing truly pervasive. The framework hinges on the concepts of delegation, arbitration and high-level service discovery.

We have built and tested a majority of the components of the proposed infrastructure.

7 Acknowledgments

The authors would like to thank Kimberle Koile and Stephen Peters for their insightful and thorough comments on this paper. We would also like to acknowledge Rattapoom Tuchinda and Gary Look as the developers of the first Hyperglue prototype.

References

[1] Portolano: An expedition into invisible computing. http://www.cse.washington.edu/portolano/.

[2] Mark Ackerman, Trevor Darell, and Daniel J. Weitzner. Privacy in context. *Human-Computer Interaction*, 16(2-4):167–176, 2001.

[3] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. In *17th ACM Symposium on Operating Systems Principles (SOSP)*, Kiawah Island, SC, December 1999.

[4] Victoria Bellotti and Keith Edwards. Intelligibility and accountability: Human considerations in context-aware systems. *Human-Computer Interaction*, 16(2-4):193–212, 2001.

[5] Jonathan Bredin, David Kotz, Daniela Rus Rajiv T. Maheswaran, Çagri Imer, and Tamer Basar. A market-based model for resource allocation in agent systems. In Franco Zambonelli, editor, *Coordination of Internet Agents*. Springer-Verlag, 2000.

[6] Adam Cheyer and Luc Julia. Multimodal maps: An agent-based approach. In *Multimodal Human-Computer Communication*, pages 111–121, 1995.

[7] Hon Wai Chun. Constraint programming in Java with JSolver. In *First International Conference and Exhibition on The Practical Application of Constraint Technologies and Logic Programming*, London, April 1999.

[8] Michael Coen, Brenton Phillips, Nimrod Warshawsky, Luke Weisman, Stephen Peters, and Peter Finin. Meeting the computational needs of intelligent environments: The Metaglue system. In *Proceedings of MANSE'99*, Dublin, Ireland, 1999.

[9] M. Dertouzos. The Oxygen project. *Scientific American*, 282(3):52–63, August 1999.

[10] Mike Esler, Jeffrey Hightower, Tom Anderson, and Gaetano Borriello. Next century challenges: Data-centric networking for invisible computing. In *Mobile Computing and Networking*, pages 256–262, 1999.

[11] Ernest J. Friedman-Hill. Jess, the Java Expert System Shell. Technical Report SAND98-8206, Sandia National Laboratories, 1997.

[12] Krzysztof Gajos. Rascal - a resource manager for multi agent systems in smart spaces. In *Proceedings of CEEMAS 2001*, 2001.

[13] Krzysztof Gajos, Luke Weisman, and Howard Shrobe. Design principles for resource management systems for intelligent spaces. In *Proceedings of The Second International Workshop on Self-Adaptive Software*, Budapest, Hungary, 2001. To appear.

[14] Christophe Le Gal, Jérôme Martin, and Guillaume Durand. SmartOffice: An intelligent and interactive environment. In *Proceedings of MANSE'99*, Dublin, Ireland, 1999.

[15] Christophe Le Gal, Jérôme Martin, Augustin Lux, and James L. Crowley. SmartOffice: Design of an intelligent environment. *IEEE Intelligent Systems*, pages 60–66, July/August 2001.

[16] Nicholas Hanssens, Ajay Kulkarni, Rattapoom Tuchinda, and Tyler Horton. Building agent-based intelligent workspaces. In *Proceedings of ABA 2002*, July 2002. To Appear.

[17] Max Van Kleek. Intelligent environments for informal public spaces: the ki/o kiosk platform. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 2003.

[18] Fabio Kon, Tomonori Yamane, Christopher K. Hess, Roy H. Campbell, and M. Dennis Mickunas. Dynamic resource management and automatic configuration of distributed component systems. In *Proceedings of the 6th USENIX Conference on Object-Oriented Technologies and Systems (COOTS'2001)*, San Antonio, Texas, 2001.

[19] V. Lesser, M. Atighetchi, B. Benyo, R. Horling, V. Anita, W. Regis, P. Thomas, S. Xuan, and Z. Zhang. The intelligent home testbed. In *In Proceedings of the Autonomy Control Software Workshop (Autonomous Agent Workshop)*, 1999.

[20] Victor Lesser, Michael Atighetechi, Brett Benyo, Bryan Horling, Anita Raja, Régis Vincent, Thomas Wagner, Ping Xuan, and Shelly X.Q. Zhang. A multi-agent system for intelligent environment control. In *Proceedings of the Third International Conference on Autonomous Agents (Agents99)*, Seattle, WA, 1999.

[21] David L. Martin, Adam J. Cheyer, and Douglas B. Moran. The Open Agent Architecture: A framework for building distributed software systems. *Applied Artificial Intelligence*, 13(1-2):91–128, January-March 1999.

[22] B. Pell, G. Christian, and P. Richard. The remote agent executive: Capabilities to support integrated robotic agents. In *Proceedings of the AAAI Spring Symposium on Integrated Robotic Architectures*, Palo Alto, CA, 1998.

[23] Brenton Phillips. Metaglue: A programming language for multi agent systems. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 1999.

[24] Manuel Roman and Roy H. Campbell. Gaia: Enabling active spaces. In *Proceedings of 9th ACM SIGOPS European Workshop*, Kolding, Denmark, September 2000.

[25] Simon Schubiger, Sergio Maffioletti, Amine Tafat-Bouzid, and Bat Hirsbrunner. Providing service in a changing ubiquitous computing environment. In *Proceedings of the Workshop on Infrastructure for Smart Devices - How to Make Ubiquity an Actuality, HUC 2000*, September 2000.

[26] Rattapoom Tuchinda. Security and privacy in the Intelligent Room. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 2002.

[27] Nimrod Warshawsky. Extending the Metaglue multi agent system. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 1999.

[28] Mark Weiser. Computer of the 21st century. *Scientific American*, 265(3):94–104, September 1991.

[29] Terry Winograd. Interaction spaces for 21st century computing. In John Carroll, editor, *Human-Computer Interaction in the New Millenium.* Addison-Wesley, 2001.

[30] Weikai Xie, Yuanchun Shi, and Guanyou Xu. Smart Classroom – an intelligent environment for tele-education. In *Proceedings of the Second Pacific-Rim Conference on Multimedia*, Beijing, China, 2001.

[31] Weikai Xie, Yuanchun Shi, Guanyou Xu, and Yuanhua Mao. Smart Platform – a software infrastructure for smart space (SISS). In *Proceedings of ICMI 2002*, Pittsburgh, PA, 2002.