Ad-hoc Composition of Pervasive Services in the PalCom Architecture

David Svensson Fors[†], Boris Magnusson[‡], Sven Gestegård Robertz[‡], Görel Hedin[‡] and Emma Nilsson-Nyman[‡]

[†]Axis Communications
Emdalavägen 14, 223 69 Lund, Sweden
davidsf@axis.com

[‡]Department of Computer Science
Lund University, 221 00 Lund, Sweden
{boris,sven,gorel,emma}@cs.lth.se

ABSTRACT

We present an architecture supporting ad-hoc composition of pervasive services, an open-source framework that implements it, and the key design principles behind it. The architecture focuses on direct human interaction, supporting combination of devices and services that are not explicitly designed to work together. The focus is on local networks, but extension is possible to wide area networks, interconnecting several local networks. The information about how services are connected and coordinated is collected in a new construct called *assemblies*. Separating this information from the services themselves allows combination of existing services in new creative ways without changing them. Assemblies can provide new services and in this way be organized hierarchically. The assembly makes the architecture of a pervasive system explicit, providing an overview understandable to users. Discovery and connections across different network technologies is supported. The architecture has been used for applications in large scale networks, and offers mechanisms useful for system integration in general.

Categories and Subject Descriptors

C.2.4 [**Computer Communication Networks**]: Distributed Systems—*Pervasive computing infrastructures and applications*; D.2.13 [**Software Engineering**]: Reusable Software—*Component based development*

General Terms

Design, Experimentation

Keywords

Pervasive systems, middleware, end-user composition, assemblies

ICPS'09, July 13–17, 2009, London, United Kingdom.

1. INTRODUCTION

In the vision of Pervasive Computing, by Weiser [35], we will be surrounded by numerous physical objects with embedded computers and communication capabilities. As this vision becomes reality, the problem of combining such devices and getting them to work together in new and creative ways becomes a demanding research area. In this paper we present an architecture, and an open-source framework, for ad-hoc composition of such devices and their services. The architecture is characterised by following mechanisms:

- There is a discovery mechanism for devices, based on a novel dynamic heartbeat mechanism.
- Devices offer services that are self-describing and can be explored interactively.
- Services can communicate over a peer-to-peer asynchronous protocol.
- The assembly construct combines services into aggregates that can be named and saved for future use.
- An assembly can mediate between services not explicitly designed to work together, and offer new services for others to use.

An assembly contains information about configuration — what services it relies on, and coordination — how the communication between these services is mediated. These aspects of pervasive systems are thus clearly separated from computation, which is provided by the services. The communication between services is supported through an interaction protocol, making services independent of the underlying network technology. Communication between devices connected to different networks is supported through routing. The current reference implementation supports IP networks, Bluetooth, and IR communication, and more technologies can be added. The architecture is based on open protocols, which means that it can be implemented in various programming languages, although the reference implementation is written in Java.

The work presented here was done as part of the *Palpable Computing* (PalCom) EU project [21, 22], which used a number of scenarios for driving the development. The scenarios were selected from professional situations, ranging from small local networks such as intensive medical care

and landscape architects in the field, to complex distributed situations, such as systems providing overview of major public events. It quickly became evident that the end-user in these scenarios could come up with uses and combinations of devices and services that the original designers could not possibly have foreseen.

We concluded that the architecture should make it possible to combine existing equipment in new ways, driven by the end users' ideas and needs, without involving the developers of the equipment. We call this *ad-hoc composition* since it allows new ways of combining services that do not need to be pre-planned. This is important since modifying services of commercial products is not feasible, and would anyway be much more technically demanding than changing an assembly. The design of the assembly mechanism has been guided by these demands.

The PalCom framework includes browsers that support exploration of devices and services, and interactive construction and modification of assemblies. This enables end users to explore new services in order to understand how they work, and to combine them in novel ways to create new assemblies. It also allows adjusting existing assemblies to changes in the set-up, such as replacing the devices used, or simply accommodating new ideas. Browsers also facilitate inspection when things do not work, down to direct interaction with individual services, in order to localize the source of the problem.

Our work is different from previous work in important ways. The idea of using a discovery mechanism, rather than a central registry, is also used for example in UPnP [31] and Zeroconf [25], but our mechanism also detects missing devices within a timeframe controllable by the user. Also, these systems cannot handle discovery and communication across different kinds of networks.

The PalCom approach includes a human in the loop when assemblies are created, who can interpret descriptions of services and directly interact with them, and mediate differences. That is in contrast to systems that rely on common standards or domain specific protocols such as Jini [34], or on ontologies and automatic reasoning, such as OWL-S [17]. Furthermore, Jini is defined in terms of a programming language, Java, and relies on a central registry for discovery.

The strict separation of configuration and coordination from computation is different from systems that require the user to embed these aspects in a programming language at the level of Java or C (as in Jini or UPnP). We share the recognition of the importance of this separation of concerns by the coordination language community [9, 2], and their reasons, support for maintainability, heterogeneity and portability are also highly desirable in pervasive computing, as pointed out in [23].

Our approach to architecture for pervasive systems has many similarities with Obje [6] (previously known as SpeakEasy). The main differences lie in our assembly construct that encapsulates configuration and coordination. Technically, Obje relies on mobile code (Java) for communicating interfaces between services, while PalCom uses XML descriptions communicated to assemblies. Our approach does not require a PalCom device to embed a JVM. It also avoids communicating mobile code, which tend to be voluminous. Furthermore, mechanisms for hierarchical composition provided by assemblies are not available in Obje.

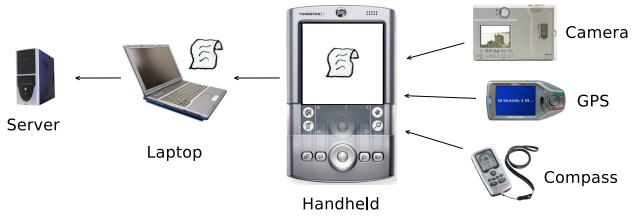

Figure 1: The GeoTagger scenario. An assembly on the PDA tags images by combining services on the camera, GPS and compass. An assembly on the laptop sends images to a server at the office.

PalCom is also different from web-service architectures. The web was designed to handle a large volume of clients, which calls for mechanisms such as central registries and client-server communication. In the pervasive systems we are addressing, peer-to-peer device discovery and communication are important to support. So, although the fields of web-services and pervasive computing address problems that appear similar at a very abstract level (and may ultimately be unified as web services become more pervasive), there are fundamental differences in both architectural considerations and the technology used.

The rest of this paper is structured as follows. Section 2 gives a motivating example of ad-hoc composition. Section 3 describes the core architecture of devices, services, assemblies, and discovery. Section 4 reports on experience from case studies of using the framework. Section 5 covers the rationale of the architecture, identifying key design principles for ad-hoc composition. Finally, Sections 6 and 7 discuss related and future work, and Section 8 concludes the paper.

2. MOTIVATING EXAMPLE

One of the end-user scenarios in the PalCom project was the GeoTagger: A landscape architect planning windmill farms is taking photos of the site, recording the position and bearing of each photo for later reference. The architect carries a GPS and a digital compass for navigation, and a digital camera for documentation. Unfortunately the devices do not work together to facilitate the situation, to tag images automatically. Thus, the architect has to record the data manually — a tedious and error-prone task.

In a pervasive computing world, the devices will provide services for taking photos, providing positions and bearings, respectively, and the architect can compose those services into a geo-tagging camera service, as well as other configurations that came up ad-hoc during the project. The PalCom prototype demonstration is illustrated in Figure 1. An assembly (shown as a scroll symbol in the figure) on a handheld performs the tagging of photos. Another assembly on a laptop uploads the tagged images to a server back at the office.

In the landscape architect scenario, there were several other situations where these devices, and a few others, were combined in different ways. The way those situations were progressively discovered, in ways that were not foreseen or planned, showed us the value of an architecture supporting ad-hoc combination of devices and services. The following section will present the concepts of the PalCom architecture introduced in this example.

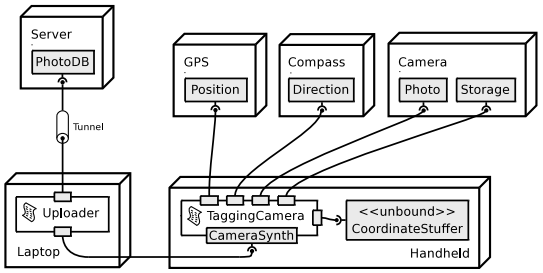

Figure 2: Assemblies in the GeoTagger scenario. *TaggingCamera* tags pictures with coordinates and directions. *Uploader* connects to a synthesized service to upload tagged images to a server at the office.

3. CORE ARCHITECTURE

The core components in the PalCom architecture are *devices*, *services*, *connections* and *assemblies*. Figure 2 shows how the example scenario GeoTagger can be realized in the PalCom architecture.

The *TaggingCamera* assembly connects to services on three devices (GPS, Compass, and Camera) and performs the actual tagging of images, using an *unbound service* (CoordinateStuffer). The resulting tagged images are offered through a new, *synthesized* service (CameraSynth). The *Uploader* assembly connects to the service of the TaggingCamera assembly as well as to a service on a database server at the office, and uploads the pictures as they become available. The figure also hints at how PalCom can span larger networks using the *tunnel* mechanism, explained in Section 5.7.

We will now examine the core concepts of the architecture, exemplified using Figure 2. See [28] for more details.

Devices are physical objects which users interact with. Typically, a device offers services that expose some aspect of its hardware functionality, such as *direction* and *position* in the example.

The identity a device has on a network is important to its users, e.g., it is important to combine pictures from *my* camera with bearings from *my* compass. Because of this, each device has a stable identity and an explicit place in the architecture. The identity of a device is independent of the addresses the device might have on the network it is currently connected to.

Services provide interaction interfaces, and are either *bound* to the device they reside on, or *unbound*. A bound service provides an interface concerning specific functionality available on a particular device, whereas unbound services provide pure computations. That means that an unbound service can be placed on any device or moved between devices as, e.g., Java components.

In the example, the bound services *Position*, *Direction*, *Photo*, and *Storage* provide interaction interfaces, while the unbound *CoordinateStuffer* provides a computation interface to manipulating the representation of an image.

Discovery of devices is performed using a broadcast heartbeat protocol. The protocol allows devices to discover other devices on the network, to connect to devices, and to request descriptors of devices and services which will be given in an XML representation[1]. In the example, the TaggingCamera assembly will connect to required services when they have been discovered and become available.

Device descriptors describe available services while service descriptors list commands a service can send or receive, along with a list of parameters for each command.

A browser developed for the architecture allows users to interactively explore services, through user interfaces automatically rendered from these service descriptions. This is useful for "remote control" and for inspection of services.

Connections are paths of communication between services. In Figure 2, connections are shown as lines between services provided by devices and assemblies. Connections can be established manually by a user, e.g. by exploring and directly interacting with a service, or automatically by an assembly.

Connections are announced explicitly by the discovery mechanism and can be viewed in a browser when inspecting the communication situation in, e.g., a room. This makes communication in a PalCom system visible to the user.

A Service Interaction Protocol defines how commands and parameters are packaged and sent between services over a network. The protocol defines message formats at a general level, i.e., not at the domain level. This means that it just allows services to send and receive messages in a common format, according to their service descriptions. This is necessary for ad-hoc composition.

Communication in the architecture is flexible in the sense that it supports different distribution forms such as *one-to-one*, *one-to-many* and *many-to-many*, but also different communication variants like message-based and streaming. Message-based communication is asynchronous and peer-to-peer. This means that there are no built-in wait states and either side can initiate messaging. A service can handle several parallel connections with different on-going "conversations" if needed.

Assemblies are the elements in the PalCom architecture that enables composition of services [26]. It can be seen as a kind of multi-connector tailored wiring, which connects to a number of services, defined by end-users or developers. An assembly specifies the following:

- *devices* it depends on,
- *services* it uses on those devices,
- *connections* it needs to those services or direct connections between those services,
- a *script* defining what actions the assembly takes upon receiving messages from the services, and
- a set of *synthesized services* i.e., services offered by the assembly.

In Figure 2, the two assemblies are shown as boxes with scroll symbols. On their borders are small boxes, for the connections to services used by the assemblies, and a service symbol with a hook for CameraSynth, the synthesized service of TaggingCamera.

Configuration is expressed in assemblies by specifying connections between services, while coordination is expressed in a script. These two aspects are in this way separated from

[1]In the small, local, ad-hoc networks of our scenarios, using broadcast heartbeats is an efficient way for devices to keep track of each other. For scaling up to larger networks, the basic protocol can be extended, as discussed in [29].

```
when image from Camera
  send stuff(image, currentCoord, currentDirection)
    to CoordinateStuffer
when taggedImage from CoordinateStuffer
  invoke photo(taggedImage)
    on CameraSynth
```

Figure 3: A part of the TaggingCamera assembly.

the computation provided by services. A service has no influence over what it will be connected to, in the same way as a device with a physical connector has no control over where a user might stick the other end of the cable. In the example, this is illustrated with the GPS position service which will send coordinates to whichever service it is connected to.

Addressing in the PalCom architecture is independent of the network technology at hand. This means that the connections an assembly depends on are established independently of where the assembly is executed. A service address identifies a particular service instance on a particular device. Due to this an assembly can be moved, without changes, to the currently most suitable device, with regards to computational resources, bandwidth, etc.

An assembly's dependency on connections can be expressed in terms of *mandatory*, *optional* or *alternative bindings*. In this way the PalCom architecture offers a fairly elaborative scheme to express dynamic behavior depending on the availability of other services.

Support for explorative composition is an essential part of the PalCom architecture. The design of the architecture is very much influenced by an explorative hands-on methodology for assembly construction. Users should easily be able to manually explore new compositions before the behavior is automated as an assembly [27].

In general, available services cannot be expected to fit perfectly together, so an assembly often has to mediate between services and coordinate the message flow. This can be expressed in the script part of an assembly.

In the example, the TaggingCamera assembly, see Figure 3, expresses that: 1) An incoming *image* from the Camera, together with the last available coordinate from the GPS, and the direction from the Compass, will be sent to the CoordinateStuffer. 2) The resulting *taggedImage* arriving from the CoordinateStuffer will be sent to whatever service connected to CameraSynth in a *photo* message. Interaction is achieved without having to change any of the involved services on the included devices.

The reference implementation of the PalCom architecture and the PalCom framework is implemented in Java. The simplest way to start to use the framework is to enable a device to provide a PalCom service. This is achieved by creating a subclass to an abstract class in the framework. In this subclass information about the service; its name and service description, needs to be added as well as behavior for incoming and outgoing commands. Bound services often use functionality from the underlying device to implement this behavior.

The framework, which observes the discovery protocol and facilitates the service interaction protocol on the available network, provides the other mechanisms needed for the device to act as a PalCom device. On a more capable device, an assembly and service manager can be included, enabling it to execute assemblies and unbound services.

The framework also provides browsers which allow users to interact with PalCom devices. The most capable one is an Eclipse based interactive assembly editor. It includes facilities to discover and explore devices and their services, as well as to edit and execute assemblies.

Assembly managers discover each other and can exchange assemblies, making it simple to move and execute an assembly (as well as unbound services) on a remote device. An assembly manager is also available as a stand alone program, which can be used to create computation servers for PalCom.

4. EXPERIENCE FROM CASE STUDIES

4.1 PalCom prototypes

Using the framework available in the PalCom reference implementation [22], PalCom teams have built a number of prototypes for user scenarios, where services have been combined into assemblies, as presented in this paper. One example is the GeoTagger prototype mentioned above.

Another scenario is Active Surfaces [4], developed at the University of Siena, involving puzzle tiles which float around in a pool. Therapists used these tiles in exercises when treating physically and mentally impaired children. Each tile runs several services and an assembly, and a group of tiles can dynamically be reconfigured to form a puzzle.

A third example is the Incubator prototype [10], also developed in Siena, where services on different devices around an incubator at a hospital were used for obtaining better integration between heterogeneous devices, and for performing cross-cutting functionality, such as alarm handling.

A more complex example is the use of the Major Incidents Overview [15] prototype which was used at the Tall Ships' Race event in Aarhus 2007 for supporting emergency response personnel; the fire brigade, police, medical teams, etc. In this scenario, developed at Aarhus University, live tracking of key persons and vessels taking part in the race was coordinated using assemblies. Information was collected as pictures from mobile phone cameras and via live streaming of video from strategically positioned video cameras.

4.2 IT in health-care

The IT-systems at a hospital is a very diverse collection of computerized functions, ranging from large systems like databases for patient records, X-ray images etc., automated laboratories or intensive care equipment with life-critical support-functions, to small single-function devices for measuring blood pressure.

Our experience from this area comes mainly from a project with Lund University Hospital (USIL) with the aim of understanding the implications of current development with increasing use of small, portable and communicating devices. During this work we have learned that hospitals of this magnitude not only face the problem of adjusting to new demands, but already have demanding problems related to structuring and re-structuring of large distributed systems.

A typical example is illustrated in Figure 4. There are over a dozen laboratories at the hospital, each with subsystems of their own, reporting results to different units, internal or external to the hospital. This system has evolved over some 30 years. Over time, new local systems have been installed to automate certain functions, often initiated from the local department. These systems have been interconnected on

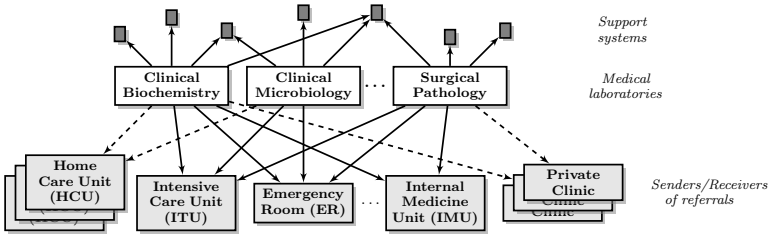

Figure 4: Example of referral handling at a hospital. Arrows show connections between units, internal and external to the hospital. Units, at the source of an arrow, know the address of the system at the other end. A solid line indicates a network connection, while a dashed line indicates use of traditional paper referrals.

a per case basis and on top of this there are connections to external services such as national registers, pharmacies, private clinics, etc. The end result is a complicated web of dependencies.

Each path of communication is managed individually by each laboratory, the protocols and formats adjusted to the receiving end. Introducing this communication has been complicated, since most of the included systems are commercial. In some cases, integration has not been feasible and results are delivered on paper and scanned for electronic storage. Changes to this structure, such as replacing a patient record system, are very complex due to the potentially large number of dependencies on other systems, some of which cannot be changed.

The IT environment is thus a complex distributed environment in itself, but also an interesting application area for pervasive systems due to the increasing use of small, portable equipment and mobile communication devices. A move towards an architecture in support of pervasive computing has the potential of also solving persistent problems related to re-structuring.

5. DESIGN PRINCIPLES

The design of the PalCom architecture was initially guided by PalCom end-user scenarios like the GeoTagger, and has evolved based on experience from additional scenarios like the one for USIL. In this section we discuss the rationale for the architecture, identifying a number of design principles that we have found essential to support ad-hoc composition.

5.1 Human in the loop vs. Automated reasoning

In managing pervasive systems one might hope that they *just work*, more or less by themselves. That would require automated reasoning, based on a common ontology and a semantic interpretation of service descriptions, along with an interpretation of a user's intentions. The result would be a fairly complicated machinery which has the risk of not being transparent to the user.

The PalCom architecture takes a less automated approach which keeps the human in the loop. Two common properties of the example scenarios are that they contain a limited number of devices and that the users have a clear view of what they want to accomplish with their composition. Therefore, our objective has been to make it easy for end-users to combine the devices and services at hand in any way they want. In the PalCom system, users can interactively explore functionality of services, combinations of services and mediations between services, and the result can be tried out directly with the actual services. In this way, the process of combining services becomes understandable, visible and light-weight. When a composition has been designed and tried out, the configuration and coordination information can be saved as an assembly. This relieves a user from having to do the same work over and over again, but also makes it possible to share configurations with other users.

Focusing on human interaction does not, however, exclude mechanisms which could help a designer find potentially useful services based on matching of, e.g., service structures or data types. A reasoning agent could make suggestions, and the human in the loop can inspect results and make sure they meet expectations.

5.2 P2P discovery vs. Central registry

Devices and services need to be found and identified in a pervasive system. In architectures designed for use in an available infrastructure, one can assume a central facility for registration and look-up. This is for example the case for Jini [34], where a central registry for services is assumed to always be available, and for UDDI [19] which offers a lookup of Web services.

In the situations we consider, that is not feasible, since devices cannot be assumed to always be available. Instead we chose a peer-to-peer model, using a broadcast heartbeat mechanism to keep all interested devices updated on which devices that are available, which new devices that have been turned on/come into reach or turned off/gone out of reach.

This kind of periodic messaging is known to be problematic if used unwisely in large networks. In PalCom the heartbeat period can be controlled by involved devices or end-users. The discovery needs vary from case to case. For instance, in an intensive care situation a doctor needs to know within seconds if a heart-rate monitor has become unavailable, while in an office environment the time-constraints for a failed device is much less pronounced.

Zeroconf [25] uses a similar approach, where devices and services announce their availability, but it lacks an efficient mechanism to report that devices are unavailable.

5.3 Self-describing services vs. Domain-specific standards

For two services to communicate they must share a common protocol at the domain level. Typical examples of such protocols are Jini's standard print service API [13], UPnP standard device categories [30] or Bluetooth profiles [3].

Figure 5: Organizing dependencies between services. (a) The traditional approach where services depend directly on each other. (b) Our approach: independent services are composed using a separate mechanism.

That approach may seem simple and straightforward, but has severe drawbacks for ad-hoc composition. First of all, a domain standard needs to be exhaustive, including all possible functionality the domain has to offer. Even for fairly small applications, like controlling the functions of a printer, that turns out to be a surprisingly complex task. In other areas, like health care [1], standards become overwhelmingly large which can be a problem when every device needs to understand all of it, or at least know enough about it to be able to ignore irrelevant parts.

Secondly, standards need to be updated when new functionality becomes available and this is usually a slow process. This might lead to different generations of equipment which, in the end, cannot communicate fully despite following the same standard.

The approach taken in PalCom is to have services describe themselves at the domain level in a human-readable format (XML). The descriptions explain what messages a service can send and receive, which parameters they have etc.

The PalCom architecture does not require domain specific standards. However, it is still possible to take advantage of available standards. The amount of work required to change from one service to another is smaller if the two services are similar, which is likely if they follow the same standard. In either case, not requiring domain specific standards makes the PalCom architecture flexible enough to handle both cases.

5.4 Separating configuration from the service implementation

In many distributed systems, a service is assumed to connect to other services it knows about. This is particularity pronounced in client-server architectures. This arrangement does, however, have the drawback that services depend directly on each other. If the service protocol is changed on one service, all the services that it communicates with need to be changed as well. Figure 5 (a) illustrates this situation, where S_2 depends directly on S_1, and needs to be changed if S_1 changes.

In the PalCom architecture, a service does not embed the address of other services it will be connected to, nor is it dependent on the protocol of other services. Instead the configuration information is stored in an assembly. The assembly connects the two services, either directly, or most commonly via the assembly itself.

Direct connections are possible in cases where there is an exact match between two services, i.e., they have corresponding message sets and no mediation is required. This is unlikely, but might occur if two services have been designed explicitly for each other. In the common case, services do not directly fit together, instead they communicate via an assembly which takes care of required mediation and coordination. This is shown in Figure 5 (b).

One of the important benefits of this architecture, with a clear separation between computation, realized in the services, and configuration and coordination, realized in the assemblies, is that re-organization can be done without having to change the services. Enabling end-users to make, at least, simple substitutions of devices or services ad-hoc in the field was one of the important requirements derived from the example scenarios.

5.5 Explicit devices vs. Device agnosticism

In many situations, such as in large Internet applications, it is tempting to focus on the services needed and ignore their location, i.e., the identity of the computers that provide them. For example, the machine actually hosting a web-service is of little importance to an end-user, in contrast to a printing service where users need to know on which printer it resides in order to collect print-outs.

In pervasive systems involving physical devices that are actually handled, the identity *is* important, and such devices cannot be substituted without a corresponding action in the physical world. In the PalCom architecture we have made it explicit that a service is provided by a device. In the health care example, if a heart-rate monitor service suddenly goes off-line, the doctor needs to know which device it was, in order to identify the patient in danger.

This approach does not exclude cases where the identity of a service provider is not important, e.g., an unbound service converting video from one format to another.

5.6 Network independent addressing

The addressing schemes of different network technologies such as IP, Bluetooth etc, address interfaces rather than devices. Devices with more than one interface can as a consequence have more than one distinct address at the same time. That happens when a device has several interfaces of different technologies, such as Ethernet and Bluetooth, at the same time, but also if it has more than one interface using the same technology. For example, IP based Ethernet interfaces for both TP cable and WLAN, which is not uncommon, or indeed two interfaces for TP cable.

Furthermore, on, e.g., IP networks, addresses are often dynamically assigned and might change from time to time, especially if a device is moved between networks. These technical observations of existing technologies clearly show that these addressing schemes cannot be used in the situations where we consider the identity of a device important.

In PalCom we have defined a separate addressing scheme where devices are assigned unique and permanent addresses. Mapping between a PalCom address and an address in a particular network is made during the discovery phase. These addresses make it possible to communicate with a device via different network paths, switching between networks as they come and go. It also enables assemblies to establish

connections to the devices that are part of its configuration, independently of how these devices can be reached. As a result, an assembly can be moved between devices and networks, without change, and still communicate with the devices included in its configuration.

5.7 Tunnels and local networks vs. The whole Internet

The initial focus of PalCom, and most pervasive systems, are local networks, such as in the GeoTagger and health care scenarios. Such simple situations are, however, often complicated by additional demands to reach remote devices, e.g. the image server at the office in the GeoTagger scenario.

Allowing access to remote devices is needed in many pervasive settings. A straightforward solution would be to consider the whole Internet as one big pervasive system. However, this is not feasible for our architecture since many of the design decisions focus on practicality in local networks and do not work on a larger scale. For example, the discovery mechanism based on broadcasting is not appropriate for the Internet.

The problem has been solved with a novel construct called *tunnels*, which connects two or more local PalCom networks. Discovery and messages between services are forwarded efficiently over such tunnels, hence, creating a larger distributed "local" network. The PalCom protocols have been designed to make this possible. For the discovery protocol, the tunnel openings on each network act as proxies for the PalCom devices on the remote network.

The discovery broadcasting mechanism is still local and only the changes in discovered devices on each network is communicated over the tunnel. The addressing mechanism for PalCom devices mentioned above makes it possible to route only the messages addressed to remote services over the tunnel.

Furthermore, the communication over the tunnels is encrypted, making it secure when transmitted over hostile networks. Tunnels differ from virtual private networks (VPNs) as a tunnel is symetrical, connecting two remote networks whereas a VPN is asymetrical, allowing a single computer to connect to a remote private network. The tunnels between networks need to be put in place explicitly by users, but once in place they can be maintained automatically.

5.8 Supporting a service interaction protocol

Many systems that support pervasive discovery, like Zeroconf, typically do not support any particular service interaction protocol: once the services have made contact, it is up to them to communicate using whatever protocol they share. Other systems, e.g. Jini, UPnP, and Web Services, offer some protocol for service interaction. However, for ad-hoc composition we have identified additional, more specific, requirements on a common service interaction protocol:

- Messages between services need to be routed over different physical networks and routed through tunnels. This requires a standardized message format that includes logical connection addresses.
- The rendering of user interfaces is based on information given in service descriptions provided via the discovery protocol. However, to actually send messages to services and to display received messages requires a common protocol for service interaction.
- Likewise, to allow an assembly to mediate between services, a common interaction protocol is required.

PalCom offers a service interaction protocol that goes together with the support for network independent addressing as well as tunnels, mentioned above. Note that the PalCom service interaction protocol is not domain-specific. It only defines the coordination messages for setting up connections, and the basic form of service messages with command name and arguments. The actual (domain-specific) commands and arguments used for a specific service are described in a service description and can be arbitrarily chosen.

5.9 Asynchronous events and P2P vs. RPC and client-server

In programming languages the notion of subroutines, procedures, functions, methods etc. have been very successful for structuring software. Execution of such constructs follows a call-return pattern, where the caller is delayed until the called construct is ready and returns.

When considering distributed systems it is tempting to re-use this successful idea, creating remote procedure calls (RPC) or remote method invocations (RMI). Programming distributed systems with such constructs appears simple and powerful since the semantic difference between a local procedure call and a remote procedure call is small and the difference in the code appears small. Technically, the differences are, however, huge [14].

When calls are remote, the time overhead of communication with a remote computer is several orders of magnitude longer than a local call. During this time the calling process is blocked and cannot proceed until the result is at hand, while during a local procedure call the machine is busy executing the procedure. Although it is in principle possible to counter this effect by introducing parallel processes, the result is often poor performance due to a large number of wait states.

Secondly, the use of RPC/RMI enforces a request-reply style of communication where one side plays the role of a client posting requests and the other the role of server, offering replies. This means that one side, the client, has the initiative to initiate communication, request-reply pairs, or pull communication. In pervasive systems this limitation is not always practical. The two sides need to communicate on a more equal footing, using both push and pull. Again, it can in principle be solved, by using two communication links for each connection, each behaving as both client and server, but this introduces new problems when synchronizing the associated services in each end.

Another successful programming construct is concurrent processes (or tasks), which communicate with events over queues. In a distributed setting with two computers actually executing in parallel and communicating over a buffered communication link this model matches the situations in our scenarios. Using this model, a process on one machine sending an event will continue with other processing and eventually receive a possible reply as a separate event. Also, this model does not favor one side over the other but communication can be initiated from both sides at any time, a mechanism often called peer-to-peer communication.

Although it may seem a very technical issue, the choice of communication metaphor has a great impact on both performance, functionality, and decoupling of services[7], which favors asynchronous peer-to-peer communication[16].

5.10 Summary of design principles

To sum up, we have identified the following important design principles for supporting ad-hoc composition:

Human in the loop to explore service functionality before automating behavior through assemblies.

Peer-to-peer discovery to allow devices to dynamically enter and exit a local network.

Self-describing services to allow combining unrelated services and to avoid dependence on out-of-date standards.

Separating configuration from service implementation to make service configurations visible to users and to allow reorganization without changing individual services.

Explicit devices because device identity is fundamental in pervasive systems where the user interacts directly with physical devices.

Network independent addressing to allow users to build compositions based on logical interconnection of devices independent on physical networking technology.

Local networks and tunnels to take advantage of the locality of the physical devices while at the same time opening the possibility to interconnect different such localities.

Supporting a service interaction protocol to allow tunnel routing, remote interaction, and assembly mediation.

Peer-to-peer asynchronous events as the basic communication mode, allowing both push and pull interaction, and to avoid the common performance problems of RPC.

The common denominator of these design principles is to base the architecture on the simple, or fundamental, case, in order to make it open-ended and extensible. Therefore, while more advanced constructs like RPC, centralized discovery, automated reasoning, etc., are complementary rather than competing, the point is that they can be added on top of an architecture not requiring them, whereas the converse is not necessarily true. For pervasive computing and ad-hoc composition, we find such open-endedness a crucial architectural quality.

6. RELATED WORK

6.1 Pervasive systems

The **Obje** project at Xerox Parc [6] has similar aims as PalCom, and has coined the term *recombinant computing* to characterize their approach for providing interoperability between devices, where a user can combine the functionality of services ad-hoc. It shares, with PalCom, the ideas of having user-in-the-loop interaction and by using self-description rather than standards for sharing protocols, but uses a technique of sending communication proxies as mobile code to communicate them. This is much heavier from most respects (communication overhead, demands on execution environment, user expertise), than communicating descriptions in XML as PalCom has chosen to do. Obje has also chosen other design alternatives than PalCom on other important aspects. Obje focuses on user-initiated direct connections between services, rather than keeping configuration and coordination in an assembly. The perspective on network independent communication, addressing and routing is also missing in this IP centric approach. In its implementation, Obje uses Jini which has directory-based discovery.

Aura [8], developed at CMU, focuses on providing automatic configuration based on interpretation of users activities/tasks. Services are discovered and descriptions of them can be communicated using XML representations in an asynchronous protocol. Services can be wrapped to conform to the APIs defined in the Aura infrastructure. Users can express their preferences as parameters in different aspects (supplier, QoS, configuration). Aura then helps the user configure the available services with help of these parameters and the task at hand. Aura shares some design choices with PalCom: discovery, supporting interaction communication, asynchronous communication, and separation of configuration information. It is more developed in the direction of automated configuration than PalCom, but seems to rely more on standards (APIs) for services, and it is unclear what support it offers for expressing coordination between services. It focuses on user involvement in configurations formed ad-hoc, but with less emphasis on understandability and visibility for the user.

Gaia [24] is a meta-operating system for active spaces, i.e., for environments such as offices and meeting rooms. The purpose of Gaia is to provide abstractions for the heterogeneous devices in such environments, and make the environments programmable. Mechanism for composition in Gaia are targeted towards developers, not users. Developers can describe context using predicates, and coordinate applications using a scripting language. There is a centralized presence service, and a central repository with information about resources. This is reasonable, because the active space has a limited and predefined physical boundary, unlike general PalCom environments. This also means that there is no issue about going beyond the local network, as with the PalCom tunnels. Gaia has been implemented using Corba, and uses its mechanisms for addressing. There are both RPC and asynchronous event between components. Gaia does not target the problem of evolving service interfaces, but assume that developers use applications currently available in the active space.

6.2 Discovery

There are several systems that only aim at providing the discovery mechanisms needed in pervasive systems.

Jini [34] from Sun uses a central repository for providing information about available services. This is a technique we have concluded to not be suitable for the small, ad-hoc networks we want to be able to support. Furthermore, the lease mechanism is designed so the provider of the service, rather than the user of a service, defines the period after which information about a non-available service is removed. This is not suitable for the situations we have encountered, where the user of a service (say a doctor using a pulse indicator) is the one who knows how urgent the situation is. Furthermore, Jini uses mobile Java code to support communication, but requires the moved proxy to implement a defined standard protocol, such as an interface for a printer. Jini thus depends on standard APIs, and all configuration and coordination needs to be embedded in Java programs, which is too static for ad-hoc combination of pervasive systems. Jini is service oriented, where the devices it resides on is not present, which also is inadequate for the pervasive systems we have encountered. It also relies on an IP-centric approach.

Zeroconf [25] supports discovery using a broadcasting technique which is well suited for pervasive systems. It does, however, lack an autonomous mechanism for removing unavailable services: they are removed when attempted to be used. Zeroconf is a service-oriented, IP-centric system, and

support for service communication, that we have found essential, is out of scope for this system.

UPnP [31] uses a broadcasting mechanism for discovery, similar in functionality to PalCom, although it misses mechanisms for tuning the frequency and thus the removal of non-available services. UPnP uses 16 standards for known devices types, and their services. It is services of these types, known in advance, that is simple to interface to. There is, however, no lightweight description language for configuration or coordination of services. Combining services thus includes a fair amount of low level programming in so called control points. There is no separation of configuration and control from services, and together this makes changes to a pervasive system not as lightweight, or with the user in control, as we have been demanding for supporting ad-hoc combinations. UPnP is IP-centric and uses SOAP over TCP which limits its use to the RMI style of communication.

DLNA [32] is built on top of UPnP for support of putting networked home audio/video equipment together, both mobile and stationary. DLNA adds some well defined refined device-types, Digital Media Players, Servers and Converters, but little more.

6.3 Web services

Web Services [33] is a collective name of Internet technologies that exhibit some of the functionality associated with pervasive systems, and can thus be compared in some respects. Discovery is here often based on central repository techniques, such as DNS for finding servers, or **UDDI** [19] for finding services. Services can be described in the Web Service Description Language (**WSDL**) [5], which matches the service descriptions in PalCom. The semantics of services can be described in **OWL-S** [17], enabling automated reasoning and selection of services. Service interaction is most commonly implemented using SOAP [11] over HTTP, and thus TCP, which enforces RPC, i.e. a synchronous, client-server style of communication. Configuration and coordination of service interaction, and the resulting aggregated functionality, can be implemented using **BPEL** [20], in plain programming, or using a more lightweight process known as *mash-ups*.

This being an IP-centric approach, there is no support for device identification or diverse network technologies. For end users of pervasive systems built this way, there would be very little possibilities for doing ad-hoc changes. Systems put together this way, using a set of fairly bulky technologies, might not be feasible to use on small devices. Such systems will be limited by the techniques involved, some of which were noted above.

There are several systems for Web services that offer techniques as the ones outlined above, for example **WebSphere** [12] from IBM and **BizTalk** [18] from Microsoft. These toolboxes, often called integration platforms, support integration in terms of business models in large IT installations. Although this is a somewhat different field, its functionality overlaps some of the aspects we have identified for pervasive systems, albeit with a difference in aim and scale. Looking closer on BizTalk, the perspective is to offer integration mechanisms for existing services, using the above diverse Web-service technologies. Integration with existing services is implemented as *adapters*, converting protocols and parameter formats, which have to be developed for each case, although there is some support for widely used technologies. Its support for business models, i.e. coordination, is realized with the language BPEL, mentioned above. These orchestrations are executed on servers becoming communication hubs. There seems to be no support for ad-hoc discovery, but it offers some dynamic binding through *ports* that enable services to provide parameters for service look-up. Development is done using the Microsoft development environments, is clearly not geared towards end users, and does not support interactive changes in place. This reflects the focus on large IT installations and organizations, rather than ad-hoc pervasive systems.

7. FUTURE WORK

The design work has been guided by the principle of introducing fundamental components of the architecture and simple mechanisms first. Through the experience of using the architecture we have, however, seen the need for extensions and additional functionality.

In several scenarios, there is a need for a dynamic binding mechanism, besides the alternative bindings, that allow services discovered for the first time to be automatically utilized by an assembly without change. Currently such situations require manual intervention.

Although being very simplistic, the Assembly language for describing coordination has proven surprisingly useful. There are, however, cases where a richer language can be motivated; e.g., where actions depend on previous events or states. The language is motivated by its simplicity, and should, however, not be allowed to grow into the full complexity of a programming language.

Security issues has not been in focus in this work. However, we do believe that state-of-the art security technology can be directly applied to our architecture. For instance, DeviceIDs are not tied to any network technology or specified in detail, so they can be cryptographically signed. The tunnels are already implemented using encrypted SSL connections. The PalCom wire protocol has support for adding signatures and authentication to messages.

Mechanisms for controlled versioning of services and assemblies have been designed into the architecture, but mechanisms for automatic update of these have only been prototyped and the details needs to be worked out.

With the ability of ad-hoc composition of services, there seems to come a frequent need to construct custom made user interfaces, tailored to a particular assembly. It would be interesting to further study how such integration and composition of user interfaces can be supported.

8. CONCLUSIONS

Ad-hoc composition means combining devices and services that were not explicitly designed to work together. We have presented a new architecture, making such ad-hoc composition possible for pervasive systems. A reference implementation for the architecture is available as an open source framework. Based on our experience from real-world scenarios and from iterative development of both the framework and prototype applications, we have identified a number of key design principles that support ad-hoc composition of pervasive services.

To match the physical world that end users interact with, devices are explicit in the architecture and networks are primarily local. Logical device identities allow abstraction

over underlying physical networks. The demands for end-user involvement have led to making devices, services, and connections interactively visible and explorable, relying on self-describing services, peer-to-peer discovery, and a service interaction protocol that is common to the services, yet domain-independent. A new light-weight construct, the *assembly*, separates configuration from computation and allows users to build and modify pervasive systems ad-hoc, without the need for changing existing services.

Acknowledgments

Much of this work was carried out as part of the PalCom EU IST project, and we are grateful to our project collegues for their work on different application scenarios, as well as on related parts of the PalCom architecture and framework. Thanks also to Thomas Forsström who implemented key parts of the PalCom networking support, including the tunnels. This work was also partially funded by VINNOVA.

9. REFERENCES

[1] American National Standards Institute (ANSI). Health Level Seven. `http://www.hl7.org/`, 2009.

[2] L. Andrade, J. L. Fiadeiro, J. Gouveia, and G. Koutsoukos. Separating computation, coordination and configuration. *Journal of Software Maintenance*, 14(5):353–369, 2002.

[3] Bluetooth.com. Specification Documents. `http://bluetooth.com/Bluetooth/Technology/Building/Specifications/`.

[4] Jeppe Brønsted, Erik Grönvall, and David Fors. Palpability Support Demonstrated. In *Embedded and Ubiquitous Computing*, volume 4808 of *LNCS*, pages 294–308. Springer, 2007.

[5] Erik Christensen et al. *Web Services Description Language (WSDL) 1.1*. W3C, March 2001.

[6] W. Keith Edwards, Mark W. Newman, Jana Z. Sedivy, and Shahram Izadi. Challenge: recombinant computing and the speakeasy approach. In *MOBICOM*, pages 279–286. ACM, 2002.

[7] P. Th. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, 35(2):114–131, June 2003.

[8] David Garlan, Dan Siewiorek, Asim Smailagic, and Peter Steenkiste. Project Aura: Toward Distraction-Free Pervasive Computing. *IEEE Pervasive Computing*, 1(2):22–31, 2002.

[9] D. Gelernter and N. Carriero. Coordination languages and their significance. *CACM*, 35(2):97–107, 1992.

[10] E. Grönvall, L. Piccini, A. Pollini, A. Rullo, and G. Andreoni. Assemblies of Heterogeneous Technologies at the Neonatal Intensive Care Unit. In *Ambient Intelligence*, volume 4794 of *LNCS*, pages 340–357. Springer, 2007.

[11] Martin Gudgin et al. *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*, April 2007.

[12] IBM. *IBM developerWorks : WebSphere*. `http://www.ibm.com/developerworks/websphere/`.

[13] Jini.org. Jini Specifications. `http://www.jini.org/w/index.php?title=Category:Jini_Specifications`.

[14] S. C. Kendall, J. Waldo, A. Wollrath, and G. Wyant. A Note on Distributed Computing. Technical Report TR-94-29, Sun Microsystems, Nov. 1994.

[15] Morten Kyng and Margit Kristensen. Supporting palpability in emergency response. In *Designing for palpability Workshop at Pervasive 2007*, Toronto, Canada, 2007.

[16] Doug Lea, Steve Vinoski, and Werner Vogels. Guest editors' introduction: Asynchronous middleware and services. *IEEE Internet Computing*, 10(1):14–17, 2006.

[17] David Martin et al. OWL-S: Semantic markup for web services. World Wide Web consortium, 2004.

[18] Microsoft. *Microsoft BizTalk Server*. `http://www.microsoft.com/biztalk/`.

[19] OASIS. *UDDI Version 3.0.2*. UDDI Spec Technical Committee Draft, Dated 20041019.

[20] OASIS. *Web Services Business Process Execution Language Version 2.0*, April 2007.

[21] PalCom Project. *Open Architecture, Deliverable 54*. December 2007.

[22] PalCom web site. Palpable Computing—a new perspective on Ambient Computing. `http://www.ist-palcom.org/`.

[23] Matthias Radestock and Susan Eisenbach. Coordination in evolving systems. In *TreDS*, volume 1161 of *LNCS*, pages 162–176. Springer, 1996.

[24] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt. A Middleware Infrastructure for Active Spaces. *IEEE Pervasive Computing*, 1(4):74–83, 2002.

[25] Daniel Steinberg and Stuart Cheshire. *Zero Configuration Networking: The Definitive Guide*. O'Reilly Media, Inc., 2005.

[26] D. Svensson, G. Hedin, and B. Magnusson. Pervasive applications through scripted assemblies of services. *IEEE Int. Conf. on Pervasive Services*, July 2007.

[27] D. Svensson and B. Magnusson. An Architecture for Migrating User Interfaces. In *NWPER'2004*, pages 31–44, Turku, Finland, Aug. 2004.

[28] David Svensson Fors. *Assemblies of Pervasive Services*. PhD thesis, Dept. of Computer Science, Lund University, Sweden, February 2009.

[29] David Svensson Fors, Boris Magnusson, Sven Gestegård Robertz, and Görel Hedin. When you're dead or gone: Undiscovery for pervasive applications. Submitted, 2009.

[30] UPnP™ Forum. UPnP™ Standards. `http://www.upnp.org/standardizeddcps/`.

[31] UPnP™ Forum. UPnP™ Device Architecture 1.0. Technical report, `http://www.upnp.org/`, Dec 2003.

[32] N. Venkitaraman. Wide-Area Media Sharing with UPnP/DLNA. *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*, pages 294–298, Jan. 2008.

[33] W3C. Web Services Architecture, February 2004.

[34] Jim Waldo. The Jini Architecture for Network-Centric Computing. *Communications of the ACM*, July 1999.

[35] Mark Weiser. The Computer for the 21st Century. *Scientific American*, 265(3):66–75, February 1991.