Composing ad-hoc applications on ad-hoc networks using MUI

David Svensson, Boris Magnusson, Görel Hedin

Lund University[1]

Abstract. The MUI framework supports composition of ad-hoc applications from services available on ad-hoc networked devices. MUI is an open-ended framework, relying on migrating user interfaces and standardized data formats for connecting services, allowing existing devices to be connected to new devices without needing any pre-defined knowledge of their services. We illustrate the benefits of the approach with scenarios involving devices like cameras and laptops that are connected through wireless networks.

1 Introduction

More and more devices in our daily environment are being equipped with wireless communication capabilities, both at home, at work, and out in the street. Using Wi-Fi, Bluetooth, and similar technologies, they can connect and form local ad-hoc networks, not relying on a central network infrastructure. This development brings us closer to the vision of ubiquitous computing [1], where computation blends into the environment, supporting people without requiring constant attention. Services can become available when needed. An example can be when a user, carrying his handheld computer, comes into the vicinity of a particular device, such as his home TV or a ticket vending machine at the train station. Services from these devices can be brought to the handheld computer at that moment. In order to make adequate use of services in this context, special preparation of the handheld must not be needed each time. Instead, services should ideally just emerge on the handheld, ready for immediate use. It should also be possible to combine previously unknown services into new applications. For a more general introduction to the challenges and goals in the field of ubiquitous computing, see for example [11]. In particular we focus on the demand for forming *ad-hoc applications*, i.e., the possibility to combine devices and services with no, or very general, prior knowledge of each other.

In order to support such ad-hoc applications we have developed the MUI framework (Migrating User Interfaces). MUI allows (1) user interfaces for services to be migrated to other devices, e.g. the handheld in the example above, making it possible to interact with the services remotely, and still in a direct fashion. Services can also (2) be connected to each other via typed data connections. Such connections can be set up re-

---

1. *Author's address:* Dept. of Computer Science, Lund University, Sweden.
E-mail: david.svensson@cs.lth.se

motely, from a third device. For example, using a handheld to connect an MP3 player to a loud speaking system.

For more complex service-to-service interactions, the user interface descriptions can (3) play a dual role of programmatic interfaces, or proxies, for the services. These proxies can be utilized by programs or scripts that glue services together in (4) assemblies.

MUI was originally started as a project with funding from VINNOVA[1], but is now also part of the EU IST project PalCom [5], which, at large, seeks to make ambient computing systems more understandable by humans. This is done by trying to meet a number of challenges, of which perhaps the most important are balancing invisibility with visibility, and finding ways of allowing construction and deconstruction of systems at appropriate levels.

This paper is structured as follows: Section 2 puts the work in context of previous work in the field. Section 3 presents a scenario that illustrates how MUI can be put to work. Section 4 gives a more in-depth discussion of the framework. Section 5 discusses the overall goals and challenges of PalCom in more detail, and evaluates the MUI framework from this perspective, providing directions for future work. Section 6 concludes the paper.

2 Previous Work

There are several earlier systems proposing solutions to the general problem of how to combine distributed services in a flexible manner. In this section we will discuss some of them and contrast them with the suggested technology in MUI.

Jini [2] is an early attempt to support combination of distributed services. The focus of Jini is programmatic, i.e. it is about programs that communicate. A central mechanism in Jini is a look-up service that aids client programs to find available services. Proxies for services are defined as Java code and in practice also the service provider is a Java program. In contrast, MUI has a user focus, i.e., it is a user that finds and combines services, at least initially. MUI uses a lightweight description of services rather than Java code which enables MUI service providers (and service customers) to be implemented in any language. This is particularly important when small service providers (such as sensors and actuators) are considered. The MUI service descriptions can be used both to directly drive user interfaces, and also as programmatic interfaces. In the latter case, glue code at the service customer will bridge from the customer to the provided service, rather than relying on standardized Java APIs that are defined and must be known prior to connecting to the service.

Speakeasy [11] and MUI share an overall idea of recombinant computing and agree on (1) keeping the user in the loop in deciding when and how components should interact with each other, and (2) using a small set of generic interfaces. Here, Speakeasy uses the terms *serendipitous integration* (the ability to integrate resources in an ad-hoc fashion), and *appropriation* (using resources in unexpected ways). Speakeasy does, howev-

1. VINNOVA - Swedish Agency for Innovation Systems, http://www.vinnova.se

er, use mobile Java code to encapsulate communication details, where MUI uses more lightweight descriptions in a textual (XML) format. For data communication, such as audio or video, the Speakeasy solution puts the burden of having a JVM also in dedicated devices such as MP3 players and speakers. The use of downloaded Java code also raises security issues as has been observed when using applets. For UI information, the use of Java to describe these means that customizing the user interface for different output devices is problematic. In contrast, the textual descriptions used in MUI allow the output devices to control the rendering. Furthermore, the MUI solution gives an architectural advantage in that the same interface description can be used both to drive a UI and to drive a programmatic API.

The focus in the Speakeasy project and MUI are partly different. The focus in Speakeasy has been on providing user interface mechanisms that enable an end user without programming expertise. This is an important aspect of MUI as well, but in addition we have a focus on building ad-hoc composite applications, assemblies, using the control part of a remote device as an API. Assemblies in MUI can offer new services which can be used in other assemblies in their turn, thus providing a hierarchical composition mechanism.

Barton et. al. [12] have chosen to build on existing HTTP technology, enhanced with a “Producer” mechanism to register services with a HTTP-server and XForms to communicate between such services and sensors (which here is used for any source of information). XForms share, with MUI, the approach to use XML-inspired textual descriptions for communication, thus avoiding dependence on Java. Being based on existing HTTP it is, however, limited by the capabilities of that technology such as a communication model based on pull and no direct support for push, as well as other restrictions.

Our early work with MUI has been presented in the master’s thesis [6] and in the paper [7]. In the master’s thesis project, a prototype with a VCR was built, where a user-interface description could be migrated from the VCR to a handheld computer via Bluetooth: the handheld computer became a remote control to the VCR. The paper [7] presented MUI’s discovery protocol, and XML-based languages for service and UI descriptions. At that stage, the focus was on migration of user interfaces. Since then we have started to work also with assemblies, and with using the interface descriptions as programmatic APIs.

3 Scenario: Distributed slideshow

As an example of a scenario where MUI can be applied, consider a slight variant of the traditional presentation session scenario, where slide shows are projected onto a large white screen. In the traditional scenario, the slide shows run on a laptop connected to the projector. When it is time for the next speaker, he either switches to his slide show, which has been copied in advance to that laptop, or he plugs in his own laptop. In our variant of this scenario, we make use of MUI to provide more flexibility. Rather than physically connecting a laptop to the projector, we use a computerized projector that the laptops can communicate with via the wireless network. Furthermore, a mobile phone can be used as a remote controller for the slide show on the laptop. This scenario is more

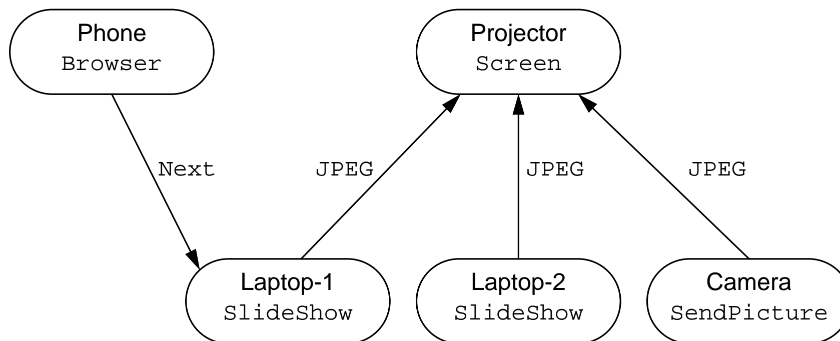

Figure 1 Distributed slideshow scenario

flexible in several ways: First, the slide shows can be run on the different speakers' own laptops, giving an obvious advantage in terms of less preparation in advance. Second, the laptops can be left anywhere in the room, and the speaker can also be located anywhere in the room, not necessarily beside the laptop. Third, more than one slideshow can be shown at the same time, with images interleaved. This can be useful in group discussions, where one person might want to jump in with a few slides in the middle of a presentation.

Figure 1 shows a set up for this scenario. The devices in this scenario: projectors, laptops, etc., are *MUI:fied*, i.e., they run the MUI system. This is easily accomplished for a laptop. The projector, on the other hand, needs to be equipped with an embedded computer with wireless capabilities. Today, this situation is easily emulated by using a standard projector and physically connecting it with a dedicated computer.

The projector has a MUI service, `Screen`, that can receive JPEG images and project them onto the physical screen. A laptop has a MUI service, `SlideShow`, which has a user interface for controlling a slide show (with buttons Play, Stop, Next, etc.), and which can send out slide show JPEG images on network connections. The mobile phone has a MUI *browser*, that can discover nearby devices and their services. Through the browser, the user can ask the `Screen` to connect itself to the `SlideShow` of a specific laptop, causing the images sent out from that laptop to appear on the screen. In the browser, the user can also ask for the `SlideShow` user interface which causes this to migrate from the laptop and pop up on the display of the phone. Then, he/she can use the phone to change slides during the presentation. The laptop also has a MUI browser, so, if desired, the user can issue the user interface commands (Play, Stop, Next, ...) and/or set up the service connections directly from the laptop as well. If several people have their slide shows connected to the projector, the latest slide is shown on the screen whenever one of them changes to a new slide.

3.1 Extending the scenario: adding a camera

The MUI system is open-ended, allowing new devices with new services to easily be added and connected. Suppose the presentation is at a conference for bottle cap collectors, and a person in the audience would like to show a particular rare bottle cap. With a camera with a MUI service `Camera` that can send JPEG images, she can simply take a picture of the bottle cap, and send it to the projector to show the image.

3.2 Ad-hoc composition

In order to support composition of ad-hoc applications, MUI relies on standardized *connection types*. This is in contrast to systems that rely on standardized service types, like Jini [2]. I.e., in MUI it is possible to connect the laptop to the projector because they send and receive JPEG images. The service `Screen` does not need any prior knowledge of the service `SlideShow`, or vice versa. This allows a service to be used in new, perhaps unforeseen, ways. The `SlideShow` can be connected to any other service that can receive JPEG images as well, e.g., printers, file storage devices, etc.

4 The MUI framework

MUI is based on services. Services are what runs on the devices, and what offer functionality to users and to other services. The services describe themselves in XML service descriptions, which are distributed to other devices on the network by means of a discovery protocol [7]. More complex services can be formed as composite services with subservices (see Figure 2), but it is the basic, atomic, services that are ultimately connected via the ad-hoc network. These have a certain type, and can be either *providers* or *customers*. We will describe below the different roles these two play in connections. Figure 3 shows a small example of an XML service description, for the `Screen` in the slideshow scenario.

The type of an atomic service determines the kind of connections that can be established to it. There are two main kinds of connections: (1) control connections, allowing the service to be controlled by another device, either programmatically or via a generated user interface; and (2) data connections, for transfer of typed data.

The Speakeasy infrastructure [11] is in many ways similar to ours: services have meta-data descriptions, and connections can be either for transmission of data, or for

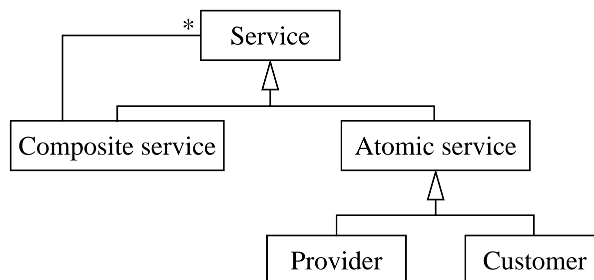

Figure 2 Service hierarchy

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE CustomerInfo SYSTEM "mui-info.dtd">
<CustomerInfo name="Screen"
              customerContentType="image/jpeg"
              urn="mui://10.0.0.3/screen">
  <Subservice urn="mui://10.0.0.3/ui"/>
</CustomerInfo>
```

Figure 3 Service description for the screen, which is a customer for JPEG images. The URN identifies the service. There is one subservice (a user interface), whose service description is referenced by its URN.

control. In both systems, there is also a browser from which the user can view and set up connections.

4.1 Control connections

The protocol implemented by a control connection is described as a service interface of the type *control*. These descriptions can be rendered as a user interface in order to allow the user to inspect the functionality of the service, and to interact with it directly, which is a key aspect of MUI. An example is the user controlling the laptop `SlideShow` service via the mobile phone: an XML description for a simplified version of this interface is shown in Figure 4.

When a control customer is connected to a control provider, the service description is migrated to the customer, and the user interface can be rendered on the receiving device. The XML description specifies mappings from actions in the user interface to what commands should actually be sent over the network to the service, and the service can also send out messages which lead to updates in the user interface. So, after the user interface has been migrated, the roles of the two sides are really symmetric—we have a peer-to-peer arrangement, where, e.g., both pull and push are possible. It is up to the service programmer, who also writes the service description, to decide upon the details of this protocol. A brief example of this kind of two-way communication will be discussed in Section 4.5.

Representing a user interface as a description has the advantage that the different browsers on different devices, having different display capabilities, can use different ways to present the user interface. This is an area that has attracted some attention in itself, see for example [8], and there are a number of XML-based user-interface markup languages, e.g. UIML [13].

An alternative use of the service descriptions is as proxies that can be used for controlling the service programmatically from another device. This allows composite services to be built, relying on distributed subservices, where a script on one device can co-

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE ControlStructure SYSTEM "mui-control.dtd">
<ControlStructure text="Slideshow">
   <InCommand id="prev" text="Previous slide"/>
   <InCommand id="next" text="Next slide"/>
</ControlStructure>
```

Figure 4 A control interface describing a simplified slideshow service. There are two commands for moving between slides.

ordinate the subservices. In PalCom, this script is referred to as an *assembly* [9]. We will briefly discuss an example of this in Section 4.5.

4.2 Data connections

Besides control connections, there are data connections for transfer of typed data. These are formed when a customer of a given type is connected to a provider of the same type. For example, the `Screen` service is a customer for type JPEG, and can be connected to providers of type JPEG, e.g., those in `SlideShow` and `SendPicture`. Currently, we use MIME types to distinguish different types.

The data connections can also be called *streaming* connections, because data flows from provider to customer, one message at a time. This suits multimedia formats, such as streaming audio and video, but can also be used in less resource-demanding applications, such as when one JPEG image is sent every time a speaker switches to the next slide in the slideshow example.

4.3 Remote connection of services

An important aspect of MUI is the possibility to connect two services from a third device. This was exemplified above, where the user connected the laptop's `SlideShow` to the projector's `Screen`, using the browser on the mobile phone. In order to allow this, there is a simple protocol which devices can use for instructing a device to connect one of its services to a service on another device: the mobile phone instructed the projector to connect its `Screen` to the laptop's `SlideShow`. Similarly, it is possible to disconnect two services that are currently connected. This functionality builds on a property of the discovery protocol, *viz.*, that devices announce information not only about the devices themselves, and about their services, but also about established connections. This support for connections gives more visibility for the user. He can see not only what devices and services there are, but he can also view and control the connections.

4.4 The MUI browser

A MUI browser has been implemented, on which nearby MUI devices and services can be inspected and controlled, and from which new connections can be established. Using the browser for remote control allows devices to be networked that do not themselves have any or very limited user interaction capabilities, e.g., sensors and actuators. Browsers can be expected to run on more resource-rich devices, such as PDAs and mo-

bile phones. The current implementation is in Java, but having an underlying JVM is not essential—the browser could be written in any language.

Existing connections can be viewed, and possibly disconnected, as mentioned above. The browser utilizes the hierarchical structure of services for making the connection process easier and more natural. E.g., the user can choose to connect the `SlideShow` service directly to the `Screen`, without opening up to see what subservices they have. In this case, there will be exactly one matching provider-customer pair, and this pair—the JPEG provider of `SlideShow` and the JPEG customer of `Screen`—will be connected. If there had been more than one matching pair, the user would have been asked to select the one he intended. This can be seen as a simple way of supporting visibility at an appropriate level.

4.5 Example of usage: the SitePack

The SitePack is one of the scenarios studied in the PalCom project (see [5], [9] for more background information). In this scenario, landscape architects out in the field make use of PalCom technology for combining devices in different set-ups, suitable for the situation at hand. One example is during the documenting phase, when photos taken at a site need to be tagged with location and other information, so they can be put together later at the office. For this purpose, the landscape architects use three devices from the SitePack: a digital camera, a GPS, and a handheld computer. When a picture is taken, the current GPS location should automatically be saved with the picture. This is realized as an ad-hoc application, with an assembly running on the handheld computer, that coordinates the camera and the GPS. The special logic needed for this particular case is in the assembly. It is important to note that the camera and the GPS are not prepared in advance for this scenario, except being PalCom-compliant at a general level.

We have implemented a simple version of this scenario using the MUI framework. The camera and the GPS expose their functionality as MUI services, both as data (GPS coordinates) and as user-interface descriptions. The user-interface descriptions are used as programmatic control interfaces by the assembly script, running at the handheld (the assembly is currently "hard-coded" in Java, but is to be written in a simpler script language later). The control interfaces are migrated to the handheld computer when the assembly is activated. As a picture is taken by the camera, the assembly gets notified through a message over the camera's control interface. In response to this, it asks for the latest picture from the camera, using an operation in the control interface. When the assembly gets the picture, the picture is tagged with the latest coordinate received from the GPS (using a special coordinate stuffer service, running on the handheld), and is sent to a back-end server for storage.

Important aspects of the implementation are that it makes use of the two-way communication that is possible with control interfaces, where both the camera service and the assembly initiate communication at different stages, and that it is an example of a user-interface description functioning as a programmatic proxy. As mentioned above, it should also be noted that only the handheld has been especially prepared for this scenario: the GPS and the camera expose their normal interfaces. The preparation of the handheld consists of construction or installation of the special coordinate stuffer service, and of writing the assembly script. The coordinate stuffer, which manipulates JPEG

image meta-data, is an example of a service which is best implemented in a full-blown programming language, such as Java, and which therefore has to be written by someone with that knowledge. It offers a service description as other services. The script, on the other hand, should be possible to write by end-users. This is where the actual adaptation to the scenario is done.

5 Evaluation and Future work

MUI involves the user in the establishment of connections between services. This gives her visibility and control over how services form ad-hoc applications. But, at the same time, this must not become a burden for her. It has to be possible also to automate the process. E.g., when she comes home, carrying her MP3 player, she might want it to automatically connect to her set of loudspeakers. Therefore, we are working on support for saving a set of connections in assemblies, which can be stored, e.g. on the MP3 player, and which can actively establish their connections. This is a simpler form of assembly than the SitePack assembly above. In our continued work, we will combine these types of assemblies into one type, so that a simple set of connections can be further customized with script logic.

From a PalCom perspective, it is interesting to look at how well MUI supports the so called *palpable qualities*, i.e. how well it meets the PalCom challenges mentioned above [5]. Our focus has been mainly on visibility/invisibility, and on construction/deconstruction. Scalability, complemented with understandability, is another important PalCom challenge. We will relate to these three in turn, and after that there will be a short discussion of security aspects, which are of course also important in the MUI context.

5.1 Visibility and invisibility

Ubiquitous computing brings a degree of invisibility to computing systems, in that they blend into the environment. PalCom highlights the need for balancing this with an appropriate degree of visibility, so that the systems remain understandable. Regarding visibility, we find it important that the user can be involved in the process of setting up connections between services. It is of course often desirable with an automatic process for this, but when the user is involved it is easier for her to understand the system. In many cases, it will also be necessary, because a program cannot be expected to understand the interfaces of the previously unknown services that will pop up in these ad-hoc networks. Another point when it comes to visibility is the merit of letting the discovery protocol distribute information about established connections, and not only about the services. This can give the user a view of the current communication.

5.2 Construction and deconstruction

For construction/deconstruction, the notion of assembly is important in PalCom. MUI combines this with limited pre-defined knowledge of service interfaces. When a new service is encountered, it should be as easy as possible for the end-user to make use of it in assemblies. We think this should be approached at different levels: At one level, it should be possible to save a set of connections as an assembly for later activation. At

another level, programmable scripts should support the need for more complex logic. In both cases, the deconstruction aspect is crucial—it must be possible to open up an assembly and inspect its parts, especially when something goes wrong.

The current implementation of MUI contains first versions of support for both levels of assemblies: it is possible to establish a number of connections and save them in a list for later re-activation, and the SitePack implementation, described in section 4.5, demonstrates a more complex assembly. Future work will involve refinement of both types, e.g. with the introduction of a script language for the more complex assemblies, and unification of the two into one concept, so they can be handled similarly.

5.3 Scalability and understandability

A third PalCom challenge that is certainly relevant for MUI is the need for supporting scalability, complemented with understandability. When using the MUI browser, the user must not be overwhelmed with the sheer amount of available services. One step in the right direction here, which we have implemented and which is also related to the visibility/invisibility challenge, is the possibility to group services as composite services. Another useful mechanism in our implementation is the use of type information for narrowing down the number of possible end-points during the establishment of a connection.

Scoping mechanisms on the network will also be needed, for making sure that the services discovered are really reachable in the current context. Similar concerns must be handled for the discovery of established connections. Ideally, only relevant connections should be shown, and for connections there is also an additional problem area of visualisation.

5.4 Security

In relation to scoping, there is the general question of security. It is important that unauthorized users or devices are not able to use your services, or spy on your connections, or modify them. To some degree, we rely on mechanisms in the lower networking layers here. In Bluetooth, e.g., two devices have to be paired before they can use each other's services. Pairing occurs once, and does not need to be done more in the future. But, there are several open issues. E.g., it has to be possible to use public services, out in the street, without having to pair each time, and still without your connections being visible to everyone. There should be some more advanced scoping mechanism also for this.

In many cases, social conventions provide sufficient security. In the slideshow scenario, all participants that have the pin codes for pairing with the devices, are also trusted with not using the technical possibilities for disturbing a presentation. Social structures could also be used for scoping. One example is the Speakeasy *converspaces* [11], where members of a converspace can invite others to share a set of components. This way, it will be possible to trust users, on the basis of trust in those who invited them. In order to use such social conventions and trust, a complement can be logging of events that can be used to find out who did what after the fact.

6 Conclusions

MUI answers some of the basic challenges in ubiquitous computing. It enables ad-hoc interaction among devices without prior knowledge of each other. They need to share a common, generic set of protocols for discovery and communicating service descriptions, but nothing that is special for a particular service. A user can very intuitively connect service descriptions to a browser and remotely control devices. It is also intuitive in a browser to connect the typed data channels between different devices and thus have them share information like audio or JPEG pictures. Here the datatypes are standardized, not the services. With this basic functionality, MUI supports many of the scenarios envisioned as ubiquitous computing.

Sets of connections can be stored as assemblies, saving the user from establishing the connection over again in case it is a situation that will occur frequently. In more complex situations, an assembly can be instrumented with a script that ties together service descriptions from remote devices (now interpreted as APIs rather than UIs). In this way complex interaction between devices can be constructed in an hierarchical fashion, thus supporting composition and decomposition.

In case an application needs algorithmic support that goes beyond what a scripting language can offer, the MUI model enables program components to be incorporated in an assembly, if only they implement the discovery protocol and offer a service description of their capabilities.

The requirement for a device to take part in a MUI system is to observe the generic set of protocols. MUI is thus an open framework that can be implemented in any language. It might be particularly interesting to implement “small” devices such as sensors or actuators in a low-level language and in such cases the effort to implement the MUI protocols should be small.

The MUI model is supporting a user-centric perspective where the user decides on when and how devices and services should interact with each other, but at the same time offers a programmatic perspective for automating tasks.

7 References

[1] M.Weiser. *The computer for the 21st century*. Scientific American, 13(2):94–10, Sept. 1991.

[2] Jim Waldo: Th*e Jini Architecture for Network-centric Computing*. Communications of the ACM, pages 76–82, July 1999.

[3] W. Keith Edwards, Mark W. Newman, Jana Z. Sedivy, Trevor F. Smith: *Supporting serendipitous integration in mobile computing environments*. Int. J. Hum.-Comput. Stud. 60(5-6): 666-700 (2004)

[4] John J. Barton, Tim Kindberg, Hui Dai, Nissanka B. Priyantha, Fahd Al-Bin-Ali: *Sensor-enhanced mobile web clients: an XForms approach*. Proceedings of the Twelfth International World Wide Web Conference, WWW 2003: 80-89, ACM.

[5] *Palpable Computing – a new perspective on Ambient Computing*. IST-002057, http://www.ist-palcom.org/

[6] Torbjörn Eklund and David Svensson. Mui: Controlling Equipment via Migrating User Interfaces. Master's thesis, Lund University, January 2003.

[7] David Svensson and Boris Magnusson. *An Architecture for Migrating User Interfaces*. In Koskimies, Lilius, Porres, and Østerbye, editors, NWPER'2004, 11th Nordic Workshop on Programming and Software Development Tools and Techniques, August 2004.

[8] Peter Rigole, Chris Vandervelpen, Kris Luyten, Yves Vandewoude, Karin Coninx and Yolande Berbers: *A Component-Based Infrastructure for Pervasive User Interaction*, International Workshop on Software Techniques for Embedded and Pervasive Systems STEPS'2005, in conjunction with Pervasive 2005, Munich, Germany, May 11, 2005.

[9] Mads Ingstrup and Klaus Marius Hansen: *Palpable Assemblies: Dynamic Service Composition for Ubiquitous Computing*. To appear at SEKE 2005, The Seventeenth International Conference on Software Engineering and Knowledge Engineering.

[10] Tim Kindberg & Armando Fox: *System Software for Ubiquitous Computing.* In IEEE Computing, Pervasive Computing, No 2, 2002.

[11] W. Keith Edwards et. al.: *Challenge: Recombinant Computing and the Speakeasy Approach*. In proceedings of ACM MOBICOM'02, Sept 23-26, Atlanta, GA, USA.

[12] John Barton et. al.: *Sensor-enhanced Mobile Web Clients: an XForms Approach*, In proceedings of ACM-WWW 2003, May 20-24, 2003, Budapest, Hungary.

[13] M. Abrams, C. Phanouriou, A. L. Batongbacal, S. M. Williams, and J. E. Shuster, J. E. (1999). *UIML: An Appliance-Independent XML User Interface Language*. WWW8 / Computer Networks, 31(11-16):1695-1708.