Abstract

We present an approach for requesting services by touching objects in the environment with a mobile phone. For example, when a photograph is shown on a mobile phone’s screen, touching a wall display with the phone triggers an operation showing the photograph on the wall display. This results in an intuitive way to interact with the environment. Furthermore, it enhances the concept of pervasive computing by mapping together the user’s physical and virtual worlds by a simple touch. The hardware that we are using for identifying the touched objects, small passive RFID (Radio Frequency Identification) tags and readers in mobile phones, will soon be available commercially. We suggest visual appearance for these tags and a general architecture that supports the development of applications and services utilizing such tags. Moreover, we present a prototype based on this architecture. The prototype software is decomposed between a Symbian mobile phone and network servers. With the prototype, a user can request services by touching objects in the environment.

1 Introduction

An increasing variety of services are currently available for mobile users. We are reaching a situation where the sheer number of services hinders their utilization. In this paper, we propose tangible interfaces as a solution to manage the service overload. Systems based on our approach will “augment the real physical world by coupling digital information to everyday physical objects and environments” (Ishii & Ullmer, 1997). As mobile phones are nowadays regularly carried and used by the users, we suggest utilizing mobile phones as components of tangible interfaces that can be used everywhere and at any time. Moreover, because the required hardware infrastructure is minimal, this approach can be used to introduce new, context-aware and pervasive services into our everyday lives. Tangible interfaces have been suggested by many researchers. To name a few, Ishii and Ullmer suggested the tangible bits approach to couple interaction with everyday physical objects and architectural surfaces (Ishii & Ullmer, 1997). Rekimoto and Ayatsuka presented several prototypes utilizing tangible interfaces (Rekimoto & Ayatsuka, 2003). Konkel et al. utilized tangible interfaces in a game prototype (Konkel, Leung, Ullmer & Hu, 2004). Want et al. linked physical documents with digital ones (Want, Fishkin, Gujar & Harrison, 1999). Swindells et al. identified objects by pointing at them (Swindells, Inkpen, Dill & Tory, 2002).

The novelty of our work lies in using a mobile phone and in a complete solution for building tangible applications. A mobile phone is equipped with an RFID tag reader, and tags are attached to objects in the environment. We developed an architecture for context-sensitive and pervasive applications that offers tangible interfaces as a general resource to request services. Furthermore, we suggest visual appearance for the tags and a model for processing the service requests to provide the desired action. Based on the architecture, we built a prototype that offers for the user a number of services that can be requested by touching objects with a mobile phone. The prototype is deployed on a mobile phone and on servers and computers in the environment. On the mobile phone side the prototype is implemented on a Symbian operating system, which is a widely used open platform for mobile devices. A native C++ implementation is lightweight and, compared to Java, provides much better performance. Furthermore, a more extensive set of system resources and functionalities are available for the native C++ implementation. The server side components are implemented in Java language.

The rest of the paper is structured as follows. The second section discusses how tangible interfaces can be utilized in requesting services. The third section describes the general context-sensitive and pervasive architecture. The fourth
section presents how objects and services can be identified, and the fifth section describes the prototype and the experiments. The sixth section concludes the paper with discussion.

2 Requesting services by touching objects

Utilization of the increasing number of services in our environments is a considerable challenge. To be able to discover all the services, to understand their details and to communicate with them requires extensive support from the underlying infrastructure. In addition to interoperability issues, service discovery is regarded as one of the most challenging tasks in the pervasive computing paradigm. Technologies like Jini (Sun Microsystems, 2005) and UDDI (OASIS, 2004) can be used to provide service discovery mechanisms, but they are meant to be used at the system level, not directly by the user. Moreover, when the user wants to discover and use a service in a new environment, the pervasive system should not confuse her by demanding complex discovery attributes and search criteria. A nomadic user prefers fast and invisible routines when interacting with the environment in such situations.

In this paper, we focus on using a mobile phone and RFID tags to employ services in the environment and to recognize the user’s actions of touching objects. Natural, calm interfaces can be obtained as a result, if the tags to be touched are intuitively associated with the services and information. Furthermore, the requirements for the system to recognize the user’s situation (i.e. context) and to reason out the proper services can be relaxed. This is because the information related to the tag and the mobile phone (e.g. location) sets major constraints on the situation. The constraints are especially strong when the reading distance of the tags is short and the reading of a tag is hence an intentional action by the user. As a consequence, a set of context-aware and pervasive applications can be built using off-the-shelf technologies. An important advantage is that the user is in control of the system. Finally, compared to completely manual operation, the user does not need to know any service parameters or attributes, as they can be set automatically when an object is touched. An example of the utilisation of surrounding services by touching objects in the user’s environment is illustrated in Figure 1 below. The black stars represent tags, and the white star represents the reader. The user has touched a laptop and a wall display. As a consequence, the user interface is extended by the keyboard of the laptop. Moreover, the content of the mobile phone’s display is shown on the wall display.

Figure 1: The user employs services in her environment by touching objects equipped with tags.
A number of prototypes have been introduced that use tangible interfaces to request services in the user’s environment. Here, ‘service’ refers to an activity that produces added value to its client – which can also be another service. In the ambientROOM (Ishii & Ullmer, 1997) prototype, moving an object representing a Web page near a speaker triggers an audio display of the Web page. The same result can be achieved by using a mobile phone – by touching a speaker with a mobile phone that has a web browser as the topmost application and the desired web page loaded in it. The Marble Answering Machine (Ishii & Ullmer, 1997) is an example of associating objects with digital entities. This machine instantiates incoming voice messages physically as marbles. The voice messages can be listened by dropping the marbles into the machine. With a mobile phone, a digital entity can be instantiated physically by touching an object. The digital entity can then be fed to an application by touching the same object again. An example of such application is the association of digital holiday pictures with a souvenir. This example illustrates how tags can be “conceptually writeable” (Want et al., 1999). Further, as suggested by Smyth et al., when the user touches an object, it could emit sounds and light associated with its place of origin (Smyth, Rajmakers & Munro, 2004). The aim would be to provide the user with a connection to past memories and places.

Want et al. described how physical documents and other objects (books, business cards, other small objects) can be linked with digital documents by tagging the objects (Want et al., 1999). A computational device detecting the tag shows the virtual document associated with the object. Furthermore, when the document is shown on the display, services can be invoked upon the document. For example, when the device senses a dictionary, it can invoke a translation service. Clearly, the device in these scenarios can be a mobile phone. The SyncTap user interface technique (Rekimoto & Ayatsuka, 2003) establishes a network connection between two devices when the user presses synchronously the special connection buttons of these devices. The precise timing of pressing the buttons needs to be recorded, and several attempts might be needed to correctly detect overlapping connection requests. When the devices to be connected are touched by a mobile phone, overlapping connection requests need no special attention, as the phone identifies the connection. Actually, touching a single device suffices in the scenarios presented by Rekimoto and Ayatsuka, where the mobile phone is the other device (replacing the PDA). For example, a document is printed by touching a printer, and a mobile phone is converted into a remote controller by touching a TV set with it.

In the Cooltown vision, people, places and things are linked to Web pages (Kindberg et al., 1996). When the user is in the physical presence of an entity, he or she can easily obtain Web content related to the entity. This vision can be realized with mobile phones and tags – to get the Web content, the user touches the entity. This is a good application, as the Web is an obvious source for information with mature tools for producing content. When the user has a Web browser as the topmost application on his or her mobile phone, it is straightforward to show the Web pages related to the touched feature. The user can create a bookmark for the Web page as well. Lots of other applications can be suggested, from checking the washing instructions of clothes by touching them to obtaining information about a piece of art in a museum. Another functionality implemented in Cooltown is the ability to transfer URLs between devices in a wireless fashion; this is called ‘eSquirt’. For example, transferring an URL from a mobile device to a projector causes the Web page to be shown on the screen. The same functionality can be achieved by touching the projector with a mobile phone that has the desired web page shown on the screen.

As noted, a number of different scenarios can be implemented by using tangible interfaces. Furthermore, regular mobile phones can be utilized for controlling the environment and for gaining access to services in the surrounding space. This leads to an optimal situation, where minimal hardware and infrastructure is needed and people can interact with the environment using the hardware they already have and are accustomed to using in their everyday life. This, however, requires some system-level support for the applications in order to make the utilization of tangible interfaces as easy as possible. We have developed a framework for processing the service requests to provide the desired action.

3 General architecture

The software architecture we have developed divides functionality into components, each of which is specialized in some task or domain area and provides corresponding functionality to the other entities in the system. Moreover, at the user level, a component can represent a service in the user’s environment, such as a printer or a projector. The components form a context-aware and pervasive middleware layer that offers a set of generic services for applications. In line with Bernstein’s definition of middleware (Bernstein, 1996), this Capnet middleware masks the complexity of networks and distributed systems and thereby allows developers to focus on application-specific
issues. Furthermore, it factors out the commonly used functions into independent components, so that they can be shared across platforms and software environments.

A substrate for components is called an engine, which is running in every Capnet-enabled device. An engine consists of core, additional and application components. Core components provide the minimum set of functionalities that are always present in an engine. Additional components extend the functionality of the engine and provide services to the other components in the system. Application components are built on top of the other components and contain the user level logic. The engines are located into the mobile terminal and the networked devices. Some examples of configurations are shown in Figure 2, where the application components are shown separately, although they are contained in the engines. A simple application consists of an application component and an engine in the mobile terminal. At the other end of the continuum, the engines in the terminal and the network form a hierarchy, the application requesting services from some engines, which in turn request services from other engines.

![Diagram of engine networks](image)

**Figure 2:** Examples of engine networks.

The Capnet component architecture is flexible and highly dynamic. Components can be searched, downloaded into the user’s device from the network and started as requested at run time. This allows the application structure to be adapted to the situation at hand – according to the services the user requests. Furthermore, location transparency enables large distributed systems to be built.

The main controlling unit of an engine is called component management. It controls all components by loading them into the system, initializing, starting and terminating them. It also processes the requests concerning component access from the other components. The service discovery component is used for discovering and locating services, resources and other components in the environment. When a component (client) needs the services of another component, service discovery tries to locate the components that offer the required service and provides a list of matches to the client. Based on that list, the client selects one component and asks component management to provide a reference to the selected component.

The messaging component provides synchronous and asynchronous communication mechanisms to the components. It delivers remote procedure calls to the destination and processes the incoming calls from other components. Messaging provides location transparency to the system, which means that the components always communicate in the same fashion regardless of whether they are located in the same engine or in engines in devices far away from each other. Messaging is based on the XML-RPC protocol, which, in turn, provides interoperability between different platforms (UserLand Software, 2003). Furthermore, XML-RPC is lightweight compared to the other alternatives, such as SOAP (W3C, 2003), Java RMI (Sun Microsystems, 2004) or mobile CORBA (Object Management Group, 2005).

Transparency is provided to the components by using specialized stub objects, which represent actual services to the local components. For example, when an application asks for some component, component management returns a component stub. The stub mediates all interaction with the application and the target component. If the target component is located in another engine, the stub uses the messaging component to deliver messages to the target component.
The core of a Capnet engine consists of the three fundamental services: component management, service discovery and messaging. The other components are optional. The user interface components create and manage application-specific graphical user interfaces on the basis of abstract descriptions provided by the applications. Abstract UIs are independent of any programming language, device type, operating system or UI toolkit. This is achieved by separating the application from its UI implementation using an XML-based UI script language to describe the abstract UI elements and their properties. The UI component renders the actual UI implementation on the target device according to the script, using available UI libraries. The UI component presents a simple interface for building and modifying the UI implementation as well as for exchanging messages and events between the application and the UI implementation.

Media components provide applications with a uniform interface for local and distributed multimedia capabilities. Context-based storage acts as ubiquitous data storage. Context components offer operations for obtaining context information, both synchronously and asynchronously, as context events. Context components, in turn, utilize the available sensor components for acquiring contextual data from the environment. Messaging can be extended by means of connectivity management that controls the network interfaces and adapts the network layer according to availability and user preferences.

The Capnet architecture can be utilized as a platform for applications that use pervasive tangible interfaces. Such a platform consists of a number of Capnet engines, one in each mobile phone, engines controlling the other devices and engines in servers offering various services to other system parts. When the user requests services by touching objects equipped with RFID tags with a mobile phone equipped with a tag reader, service requests can be processed as illustrated in Figure 3: When the application is started, it subscribes context events from a context component (1) which, in turn, subscribes RFID events from a sensor component (2). When the user touches an object, the sensor component sends an event to the context component (3). The context component fetches from a database additional information related to the RFID tag and the touched object (4). Based on the available information, the context component recognizes the user’s context and sends an event to the application, if the recognized context matches with the event subscription (5). For example, a text editor application could be informed when the user touches a printer. It could then print the open document that the user is editing.

![Figure 3: Processing service requests.](image)

### 4 Identifying objects and services

When the aim is to request services by touching objects, a solution is needed to communicate to the user which objects can be touched, and what services can be requested in such a way. Furthermore, when the user touches an object, information needs to be communicated to the system offering the services. Preferably, the information read from a tag should be in a standard form, so that the same tags could be utilized by many different devices and services. In the Capnet architecture, objects and services are identified by using the EPC code representation currently being standardized (EPCglobal, 2005). Each RFID tag stores an EPC code that is communicated to the RFID tag reader when the user touches a tag. This code is used to fetch information about the object and the services it represents. An EPC code defines the serial number and the class of the object. Furthermore, it determines the
organization responsible for maintaining object classes and serial numbers and possibly also an action to perform when the object is touched. We are using a 256-bit set consisting of the fields shown in Table 1.

**Table 1:** EPC code fields and the number of bits reserved for each field

<table>
<thead>
<tr>
<th>Field</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>8</td>
</tr>
<tr>
<td>Filter value</td>
<td>4</td>
</tr>
<tr>
<td>Organization prefix</td>
<td>64</td>
</tr>
<tr>
<td>Object class</td>
<td>56</td>
</tr>
<tr>
<td>Serial number</td>
<td>124</td>
</tr>
</tbody>
</table>

The header field specifies the format of the following bits. We use the filter value to define the action to be triggered when the tag is touched. For example, the binary filter value 0001 on a printer tag can be associated with the action *Print*. Another value might be associated with the action *Call maintenance* – touching such a tag could automatically initiate a call to the administrator of the printer. The organization prefix defines the organization in a globally unique way. University of Oulu, Department of Electronic Engineering, for instance, has the unique prefix 00370245895545000. Object class determines the type of the object (i.e. the entity the tag is associated with), for example, *Printer, Poster, Room* and *Doorway*. Finally, serial numbers enumerate all the different instances of each object class. By object class and serial number, we can uniquely identify all objects and associate services with either object instances or object types. When a tag has been read, the EPC code delivered by the sensor component can be used as a keyword in querying additional information related to the object. However, the first prototype reported in this paper does not yet contain this feature, since all information needed by the implemented services is included in the EPC codes.

Each object class in our identification framework has one or more actions associated with it. Each tag attached to an object, in turn, has one of the possible actions described in the EPC code stored in the tag. This action is communicated to the user by the visual appearance of the tag. When the user touches an object, the filter value is delivered to the system as part of the EPC code. The default action (filter value 0000) is to list all actions related to the object and to let the user choose which one to use. This is referred to as a general action. Other actions are specific to object classes - each class can have seven additional actions at the most.

For instance, a printer object class in our prototype has three additional actions: *Print, Contact maintenance* and *Info*. Thus, a printer can have a maximum of four tags attached to it. Each tag has a different visual appearance. The codes stored in the tags are otherwise identical, but the filter values specify different actions. Touching an *Info* tag causes printer information to be shown on the mobile phone’s display, and touching a general tag causes the three additional actions to be listed to the user.

Examples of different object classes and actions are illustrated in Table 2 below. The context component uses object class and filter values to match the subscribed context events with the RFID events (See Figure 3). For example, if an application has subscribed the *Printing service requested* context event, the object class stored in the touched tag is 0011 and the filter value is 0001, matching is accomplished successfully, and an event is delivered to the application.

**Table 2:** Examples of object classes and filter values

<table>
<thead>
<tr>
<th>Name</th>
<th>Object class</th>
<th>Printer actions</th>
<th>Filter value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(for future use)</td>
<td>0000 (0)</td>
<td>General</td>
<td>0000 (0)</td>
</tr>
<tr>
<td>Doorway</td>
<td>0001 (1)</td>
<td>Print</td>
<td>0001 (1)</td>
</tr>
<tr>
<td>Room</td>
<td>0010 (2)</td>
<td>Contact maintenance</td>
<td>0010 (2)</td>
</tr>
<tr>
<td>Printer</td>
<td>0011 (3)</td>
<td>Info</td>
<td>0011 (3)</td>
</tr>
<tr>
<td>Brochure</td>
<td>0100 (4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Computer</td>
<td>0101 (5)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Projector</td>
<td>0110 (6)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Media player</td>
<td>0111 (7)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Television</td>
<td>1000 (8)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Curtains</td>
<td>1001 (9)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lights</td>
<td>1010 (10)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Doorway actions</th>
<th>Filter value</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>0000 (0)</td>
</tr>
<tr>
<td>Map</td>
<td>0001 (1)</td>
</tr>
<tr>
<td>Profile</td>
<td>0010 (2)</td>
</tr>
<tr>
<td>Info</td>
<td>0011 (3)</td>
</tr>
</tbody>
</table>
Having multiple tags associated with the object reduces effectively the activity needed from the user when interacting with the environment. If several services can be associated with the object, the user can touch directly the tag triggering the desired service instead of selecting it from a list of all available services. Furthermore, if the visual appearance of the tag is intuitively associated with the corresponding service, requesting the service is an easy operation. Our goal is that the user can request services without interrupting his or her activities to find out how to perform the request. In an optimal case, after requesting a service, the user can smoothly continue whatever she or he was doing, only supported by the requested service.

To achieve this goal, we have defined the visual appearances for the tags (visual tags) used in the experiments, see Figure 4. The visual tags were designed to be associative with the action that is stored as part of the EPC code in the tag. The general tag is an exception, as it is used as a part of the other tags – they have the appearance of the general tag augmented by an action symbol. The call tag, for instance, has an earpiece symbol.

Figure 4: Visual tags.

5 Prototype and experiments

Based on the architecture, we have implemented a prototype which allows services to be requested by touching corresponding objects in the user’s environment. The prototype offers for applications a general framework for handling such service requests. It is deployed as a single Capnet engine on a Symbian OS mobile phone and several Capnet engines on the network servers hosting the service components. The phone is equipped with an external RFID reader, which communicates with the engine. As we are currently polishing the EPC code representation, the RFID tags do not contain full EPC codes but data sufficient to identify the objects and the services. Future prototypes will contain the reader integrated in the phone and full EPC codes stored in the RFID tags.

Services and resources in the user’s environment are represented by Capnet components. These components are physically running on the Capnet engines in the network and are available for other components or users. Virtually all components are running in the same address space regardless of their physical location. Applications in a mobile device can, for example, start using a display component that is running on a desktop computer. Since the architecture handles component interactions in a location-transparent manner, the application sees the display component as a local resource. A stub represents the real service to the application and mediates all needed interaction between the application and the service components (see Figure 5).

As resource-limited mobile devices cannot be overloaded with heavy software infrastructures, the phone’s engine contains only a minimum set of components and primitive functionality. It contains component management for handling and serving the components in the local device. The messaging component enables the components in the mobile phone to communicate and co-operate with each other and with other components in the environment. Although service discovery belongs to the core, it is not required in our prototype, since the user can discover services by physically touching them with a device. Touching provides the needed service parameters and attributes required for connecting to and utilizing the desired service. In other words, when the user touches objects, new local services are discovered.
In addition to the core components, the mobile device hosts a context component, which produces contextual information, such as calendar information. Moreover, the RFID sensor offers RFID events to the context component. The RFID reader is like any other sensor device used by the context component. Applications can utilize the context component by subscribing different types of context events from it.

A Nokia 6600 mobile phone was used in the experiments. The software (i.e. engine) in the mobile phone is implemented entirely on top of a Symbian operating system in a native C++ language. Components in a mobile phone are implemented as polymorphic DLLs, which also package the components in individual entities. A polymorphic DLL has only one exported function, which returns an object implementing a well-known interface. This interface is shared by all Capnet components. Polymorphic DLLs enable component management to handle components as atomic items. They can be downloaded from the network and integrated into the system at runtime as needed. The RFID reader software is a separate application that communicates with the prototype using the messaging capabilities of the Symbian operating system. The fingerprint of the middleware architecture implementation is less than 50 kilobytes, and the whole prototype requires about 65 kilobytes of memory.

Here we report the performed experiments in the form of a scenario. The prototype produced successfully the functionality described in the following scenario: Kari is a new employee of a company, which utilizes the Capnet system in their office space. All services in the workplace are tagged with RFID tags, and they can be used by touching them with a mobile phone. Services include office equipment (printers etc.) but can be also fully virtual. Map service, for instance, offers a map of the current floor.

The scenario starts when Kari enters his new workplace. At first, he has a meeting with his manager Timo. However, Kari finds out that Timo is not in his office at the appointed time. Kari has been told that the building is equipped with services, which can be accessed by touching them with a mobile phone. Services are indicated with tags that can be easily recognized from the environment. Moreover, the visual appearance of each tag suggests in an intuitive fashion the service offered by it. Kari notices some tags next to the door of Timo’s office, and one of them looks like a calendar. Intuitively, Kari associates the calendar tag to Timo’s schedule and touches it with his phone. This action opens a calendar application with Timo’s schedule on Kari’s phone’s display. It appears that Timo has scheduled the meeting to a meeting room instead of his office.
While figuring out the way to the meeting room, Kari notices a map tag next to the calendar tag. As Kari touches it with his phone, the phone opens an application, which shows a route to the meeting room and indicates that Timo is in the meeting room. There is one more tag next to Timo’s office labelled as a general tag. It would have caused a list of all available actions to be shown on the screen of Kari’s phone. Kari could have accessed Timo’s calendar and location from that menu as well.

As Kari starts walking towards the meeting room, he notices a printer in the corridor. Kari finds out that the printer is labelled with two tags, the first one looks exactly the same as the one by Timo’s door. Kari associates the second tag to a printing action and touches it with his phone. Since the map application is on top of the phone’s display, the map is printed. As a curiosity, Kari also touches the first tag. This causes a list of available services to be shown on his phone: Print, Info and Call maintenance. Thus, Kari finds out that with a general tag he can access all available services. Finally, Kari finds the meeting room. When he is about to enter the room, he discovers some tags next to door of the meeting room. He associates one tag with the meeting profile and touches it with his phone. As a consequence, the phone changes the profile into the meeting mode, and no phone calls disturb him during the meeting. (As Symbian OS does not allow us to change the profile programmatically, we could not fully implement this feature. Hence, profile change is indicated for the test user with a pop-up window.)

When the meeting is over, Kari needs to make a reservation for a cocktail party for the evening. He finds some tags from the brochure of the company organizing the party. One tag is next to a contact number. Kari associates that tag with a phone call and touches it. As a consequence, a phone call is initialized, and Kari makes the reservation. Finally, before going back home, Kari finds out that he has to enroll on a weekend course. While he is writing an SMS to the course assistant, he notices a computer with a tag attached on it. Touching the tag enhances the mobile phone with advanced input and output capabilities: the display of the SMS application is transferred on the computer’s larger screen, and the text can be typed using the computer’s keyboard.

Some stages of the experiments are illustrated in figure 6 below. In the pictures, the user is holding the tag reader that communicates via Bluetooth with the mobile phone. In the situation shown on the left, the user prints a document. Next, the user locates another employee. In the third situation, the user is at the office doorway and uses the services he found. In the situation shown on the right, the user interface of the SMS application has been transferred on the computer’s screen.

6 Discussion

We have created a framework for utilizing services in the user’s environment by touching objects. This framework is utilized in the Capnet architecture that supports the development of context-aware and pervasive applications. We have implemented a prototype based on the architecture and created a test environment with a number of real services. The prototype consists of software in a mobile phone and services in the environment, which can be accessed by the user by touching objects with the phone. The visual appearance of the RFID tags attached to the objects is designed to be associative with the services that can be requested. Furthermore, the objects and the services are identified by an EPC code read from the RFID tag. Selecting the EPC coding standard facilitates the use of a single set of tags for all devices and services. Standard development work is currently active, and we are closely following the upcoming releases.
The visual appearance and the EPC code enable services to be activated by a single touch, as the user only needs to select the right tag, and all information required in service activation can be accessed using the EPC code. Interaction with the environment is easy, since the user knows what to expect when a tag is touched. A further advantage is that the user is in control of the system.

We have tested the prototype in several scenarios where the user utilizes ambient services in different situations. The experiments verify that tangible interfaces can be implemented with the current mobile phones. Although, in the first prototype, the reader was a separate device, the Capnet middleware for mobile phones can be used without changes in the new models with integrated RFID tag readers. Such phone models, the Capnet middleware and tags attached to the environment form a complete solution for context-aware and pervasive services utilizing tangible interfaces. Future work includes full integration of EPC codes into the Capnet architecture – in the first prototype the RFID tags did not contain complete EPC codes, and the Capnet database was not utilized. We will study user experiences in the near future in order to find out how well users associate the visual tags with the relevant services. The visual tags will be developed based on the results.

Acknowledgments

This work was funded by the National Technology Agency of Finland. The authors would like to thank all the personnel in the Capnet program and the participating companies. Furthermore, Marketta Heinonen and Heikki Laaksamo are acknowledged for designing the visual tags and specifying the EPC code representation, respectively.

References


