Palpability support demonstrated

Jeppe Brønsted[1], Erik Grönvall[2], and David Fors[3]

[1] jb@daimi.au.dk
Dept. of Computer Science, University of Aarhus
Aabogade 34, 8200 Aarhus N, Denmark
[2] Communication Science Department, University of Siena
Via Roma 56, 53100 Siena, Italy
[3] Dept. of Computer Science, Lund University
Ole Römers väg 3, 223 63 Lund, Sweden

Abstract. In ubiquitous computing, as more and more devices are embedded into the environment, there is a risk that the user loses the understanding of the system. In normal use this is not always a problem, but when breakdowns occur it is crucial that the user understands the system to be able to handle the situation. The concept of palpable computing, introduced by the PalCom project, denotes systems which support such understandability. In PalCom, a set of prototype scenarios provide input for an open software architecture and a conceptual framework for palpable computing. One of these prototype scenarios is based on the Active Surfaces concept in which therapists rehabilitate physically and mentally impaired children by means of an activity that stimulates the children both physically and cognitively.
In this paper we demonstrate how palpability can be supported in a prototype of the Active Surfaces. Services on the tiles have been developed using the PalCom service framework that allows them to be combined into PalCom assemblies. The support for palpability is shown by examples of use scenarios from the work of the therapist who can inspect and alter the runtime state of the tiles to change their configuration and cope with breakdown situations. The prototype implementation runs on a standard PC simulating the network layer and a first reference implementation has been made on the target embedded platform.

Key words: Palpable computing, ubiquitous computing, middleware, services, assemblies, inspection, simulation framework

1 Introduction

Palpable computing is a new perspective on ubiquitous computing, in which traditional ubiquitous computing challenges such as invisibility and composition are complemented with visibility and deconstruction. The concept has been formed based on the observation that when applied in real settings, ubiquitous computing systems tend to become hard to understand for users. Mark Weiser painted a vision of systems being 'physically invisible' and that these systems

also mentally (as maybe physically) could disappear through use. Dr. Weiser describes the concept well as follows; "Whenever people learn something sufficiently well they cease to be aware of it." and "The most profound technologies are those that disappear" [1]. This implies not only that as we learn to master a technology, we move the use (or perception) of it from a foreground to a background cue [2], but also that a technology that has the capacity to allow the user to interact with or through it as a background process is a more thoughtful, intense or reflective technology.

As part of the ubiquitous conceptual framework and closely related to the work regarding foreground and background cues is the notion of 'calm technology' [3]. Calm technology as described by Weiser and Brown regards how technology can move from the centre of our attention out to the periphery, and between those two states as required by the situation at hand. The vision of calm technology is that technology should not overload us with information or require an ongoing 'active' mental activity. Weiser and Brown argues that this can be reached in two ways; 1) Allowing the technology (or information) to move between the centre to the periphery of our attention (and between these two states) and 2) by enhancing our peripheral reach. This is done by allowing more data to enter the periphery cues. As described in their paper, a video conference could be an example of technology that enhance the peripheral reach in respect to an ordinary telephone call where the users cannot use facial and body expressions as part of their communication [3].

In many ways, ubiquitous systems tries to embed the notion of 'ready at hand', meaning that these highly distributed, networked systems and devices should adapt to the current needs of the user or users. In normal use the system should be invisible and not interfere with the present task. However, when a breakdown occurs the user should be able to inspect the system to determine the reason and, if possible, resolve the situation. If the system is composed of multiple devices, it should be possible to replace malfunctioning devices with new ones without having to recompile or restart the system. We do *not* claim that techniques such as self-reconfiguration, error detection and fault tolerance should not be used. We make the observation that such mechanisms will never be perfect and that we therefore, as a supplement, need a way to handle the situations where the mechanisms are imperfect.

Palpable computing is researched in the EU IST project PalCom [4]. The main output of the project is a conceptual framework for palpable computing, an open architecture supporting palpable computing, and a collection of tools to be used in development of palpable computing applications. A part of the work in the project deals with developing palpable computing prototypes using participatory design to provide input to the conceptual framework and the design of the open architecture.

The *Active Surfaces* [5] concept provides support for physical-functional and cognitive rehabilitation in a swimming pool setting. The concept has been developed using participatory design techniques in corporation with therapists and patients. Through analysis of the rehabilitation practice an activity (i.e. a num-

ber of different games) has been developed in which children assemble floating tiles into meaningful configurations. Each of the tiles is a resource constrained embedded system that communicates using only a low bandwidth short-range infrared link. The only output available to the user is a set of light emitting diodes and therefore the game is an example of a ubiquitous computing system where it is essential that the physical and functional characteristics are such that palpability can emerge during use.

For the software on the tiles, support for palpability is achieved by adhering to the PalCom open architecture [6], and by building on the PalCom service framework [7]. Services developed using the framework can be combined into PalCom assemblies, which coordinate the services and provide support for inspection, deconstruction and reconstruction. Through interaction with the assemblies, the therapist can inspect and change the configurations of the tiles. This way, she can adapt the therapeutic activity in the middle of an exercise, and the visibility given by the assemblies helps her cope with unexpected breakdown situations.

The rest of the paper is structured as follows. In the next section we describe the Active Surfaces concept and the physical and functional aspects of the prototype. Section 3 presents the PalCom software architecture and demonstrates how it can be used to support palpability in the implementation of the prototype. In Section 4 scenarios from therapist work are presented, together with an evaluation of how the prototype supports palpable qualities. Section 5 sums up conclusions and presents future work.

2 Active Surfaces

Active Surfaces is a concept developed for rehabilitation practitioners being a support for physical-functional and cognitive rehabilitation treatments in a swimming pool setting. Therapists working in cognitive and physical rehabilitation with disabled patients usually experience their job as challenging and demanding. Every time the therapist starts a treatment she has to define a specific program and ad hoc solutions with the aim of designing a rehabilitation intervention that could adapt to the individual patients' needs. Thus, the work of the therapist is mainly characterised by creativity both in designing engaging activities and suitable tools.

The lack of integration of physical and cognitive rehabilitation represents a constraint for current rehabilitation practice. The cognitive tasks are usually too static and children may lose attention. On the other hand, motor rehabilitation is very demanding at a physical level and is based on repetitive sequences of actions: patients often perceive them as tiring and not engaging. Here the Active Surfaces allow an integration of these two therapeutic goals with the activity. Water as such is an interesting context from the activity perspective; Water creates a safe context where impaired people can move autonomously relying on the added support to the body, something they cannot do elsewhere. Apart from this, water also poses specific and interesting research issues both for the development of digital technologies and for the therapeutic practice.

The work has been driven following a co-evolutionary method [8, 9]. The approach integrates participatory design with creative concept design, using different typologies of scenarios for converging ideas into solutions. The concept and project has been developed together with children affected by different impairments undergoing therapeutic activities in the swimming pool, their parents, trainers and therapists at the swimming pool and at the 'Le Scotte' hospital in Siena, Italy. The early phases of the fieldwork have been devoted to understand the activity, to define requirements, and to collect best practices. On this basis, the concept of the Active Surfaces has been developed, capitalising on participatory design activities and creative workshops together with Travelling Architect [10] and Future Lab [11] sessions.

2.1 The Prototype

The prototype consists of a set of floating tiles (figure 1) that can be connected to each other to form a network. The tiles support multiple games by having a simple composable physical appearance and multipurpose programmable hardware. On each of the tiles' four sides magnets are placed to make the tiles "snap" together when they are in close vicinity. On the top of the tile is a replaceable plastic cover also held in place by magnets. The image on the cover depends on the game. On each side of the tiles light emitting diodes (LEDs) provide visual feedback to the user.

Fig. 1: Tiles

Inside each tile an embedded system uses infrared light to communicate with and detect the presence of other tiles. Two tiles can only communicate if they are close to each other. Figure 2 shows an overview of the hardware components in the tiles. The main computational unit is the UNC20 module, which is an ARM7-based embedded system running uClinux[12] at 55MHz with approximately 8MB ram. The UNC20 module communicates with a sideboard using a serial connection. The sideboard is responsible for controlling the infrared communication and the LEDs. The bandwidth of the infrared communication is approximately 600 bits per second.

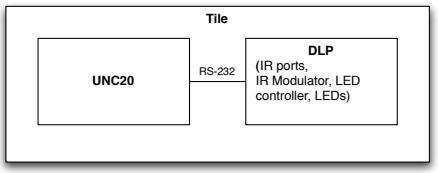

Fig. 2: Hardware

2.2 Games

The tiles support multiple games and in the following we describe a few suggestions. To change the current game the therapist connects the tile to a PDA

running PalCom software. Since the PDA is not suited for a wet environment this should be done prior to the training activity.

A lot of games can be imagined for the Active Surfaces. However, for a game to be appropriate for the tiles it should support both physical and cognitive rehabilitation while at the same time be implementable on the resource-constrained devices. Furthermore, to be able to help a wide range of patients the set of games should be of varying difficulty, both on the physical and on the cognitive level. Finally, the games should be open ended and configurable so that they can be adapted and customised to each rehabilitation session.

In this section we describe three games with different properties with respect to physical and cognitive rehabilitation. The first game, *catch*, is meant to only require simple cognitive effort but challenges the patients reflexes, speed, and coordination. The second game, *scrabble*, has the requirement that the patient should be able to form words out of letters. The last game, *puzzle*, is a traditional puzzle game in which an image is created by assembling the tiles in a specific pattern.

Catch In the catch game the therapist aligns a set of tiles and gives another tile to the patient (at the bottom in figure 3). When the game is started the point of the game is for the patient to try to catch the light by approaching her tile to the glowing tile within a certain timeframe. If she succeeds another random tile will light up (not hers) and she tries to catch that one. When she eventually fails to catch the light before the time limit her tile will blink how many lights she caught. The game can be adapted to the patient by configuring the length of the timeframe.

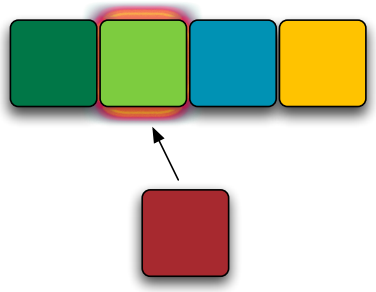

Fig. 3: Catch

Scrabble In the scrabble game each tile has a letter on the surface. The patient uses the tiles to create words. When a tile is part of a word it lights up on all four sides. Each tile should be aware of what letter it has on the face. The memory requirement for the game depends on the number of tiles and on which letters the tiles have. As an example at least 24 English words can be generated from letters on the tiles in figure 4[4]. Since this number grows exponentially with the number of tiles it is not feasible to store all possible combinations on each tile. Instead, only the valid words for a *particular* tile-letter configuration should be uploaded to the tiles. The letter configuration should therefore be uploaded along with the game before the training activity.

[4] a, ad, at, act, arc, art, cad, car, cat, had, hat, rat, tad, tar, arch, card, cart, char, chat, dart, hard, hart, chard, and chart

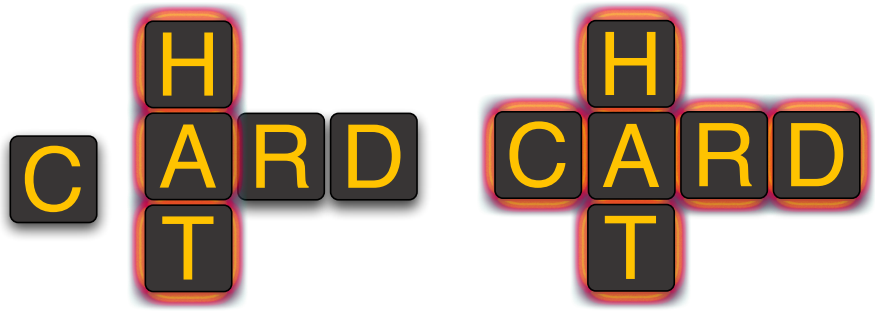

Fig. 4: Scrabble

Puzzle In the puzzle game the face of each tile is part of a larger image (see figure 5). Initially the tiles are spread in a random pattern after which the patient starts to solve the puzzle. As the game progresses the patient gets continuous feedback from the LEDs. When two tiles are connected correctly the corresponding sides light up (fig. 5a). When all of a tile's neighbours are correct all sides of that tile light up (fig. 5b), and finally when the puzzle is solved the outline of the solution lights up (fig. 5c).

(a) (b) (c)

Fig. 5: Puzzle

During the session the therapist can change the faces of the tiles to make a new puzzle. To reprogram the tiles a special *assembler tile* is used. The assembler tile has the same physical appearance as the other tiles, but also has a button. To make the tiles remember the new solution they are arranged in the solution pattern and the assembler tile is put next to one of the tiles and the button is pressed. After this, the tiles will remember the new solution and can be scattered randomly again. This way of programming the tiles by showing them the correct solution has some similarities with *programming by example* [13] and *physical programming* [14]. The LED feedback can be configured by the therapist to alter the difficulty level of the game. It is, e.g., easier to solve the puzzle if all the outer edges of the final solution will light up as the game is started.

The different game types described above all have different game rules. These rules defines the base of a game. Apart from them, different behaviour can be configured to support the game rules and the activity. This can for example be

different output to the end-user to aid in accomplishing the task, i.e. the game. This configuration can be physical and logical.

3 Implementation

In this section we demonstrate how the PalCom software architecture and runtime system can be used to implement the Active Surfaces prototype in a way that supports palpability. We describe the PalCom runtime system (section 3.1) and a simulation framework that can be used to experiment with the tiles on a standard PC (section 3.2). The top layer of the runtime system is the application layer in which applications are built by composing services (section 3.3) into assemblies (section 3.4). Finally, in section 3.5, the implementation of the puzzle game is described.

3.1 PalCom Runtime System

The PalCom runtime system (see figure 6) consists of four layers: the execution platform, the runtime environment, the middleware management layer, and the application layer. The execution platform consists of hardware and optionally an operating system. Presently multiple hardware platforms are supported including the UNC20 [15] and standard PCs. The runtime environment consists of standard libraries and a runtime engine which can be Sun's Java VM or the Pal-VM [16], which is a compact virtual machine specially designed for embedded systems for ubiquitous computing. If the hardware platform is the UNC20 only the Pal-VM is supported. The middleware management layer consists of managers handling resources, services, assemblies, and contingencies. For further description of the middleware managers we refer to [6]. At the time of writing, the memory footprint of the middleware management layer is too big to fit into the memory of the UNC20 (app. 8MB). Therefore, concurrently with the development of the hardware for the tiles and the optimisation of the middleware management layer, the software for the tiles has been developed to run on a standard PC with simulated infrared communication, on top of Sun's Java VM. When the middleware management layer fits into the memory of the UNC20, the implementation of the prototype should be able to run unaltered on the UNC20.

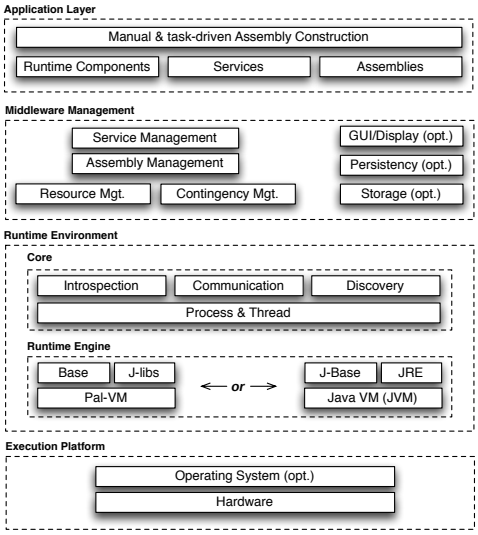

Fig. 6: PalCom layered runtime system

3.2 Simulation Framework

To ease the development of game logic and software for the tiles a simulation framework (figure 7) has been developed. Having a simulator available makes it possible to develop software and hardware in parallel and high level tools that are not available for the embedded platform can be used for debugging and profiling. Furthermore, testing involving repeated rearrangement of the tiles is much easier done using a mouse in a graphical user interface than physically moving the actual tiles around.

The simulator consists of a model of the swimming pool as a medium for infrared communication and a graphical user interface for manipulating the physical location of the tiles. The user interface is connected with the pool model so that when a tile is moved in the user interface, the pool model is updated accordingly.

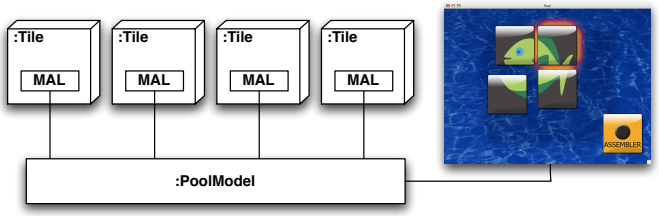

Fig. 7: Simulation framework

The model of the pool is also used in the medium abstraction layer of each of the tiles. When a tile sends a message, the medium abstraction layer of the tile accesses the pool model to determine which tiles the tile is connected to (if any) and delivers the message accordingly. From an application developer's perspective it is transparent whether the simulation framework or the physical hardware is used. The only part of the middleware that interacts with the simulation framework is the media abstraction layer and therefore system behaviour experienced on the simulator is likely to be similar on the embedded platform.

3.3 Services

The software implementing the functionality of the tiles is divided into services using the PalCom service framework [7]. As described in [6] a PalCom service is an entity that 1) contains a number of runtime components, 2) is discoverable, and 3) can be invoked remotely. The interaction with the service is done using an explicitly defined service interface. A PalCom service has a set of commands, in-going or out-going, that optionally can be grouped. An in-going command specifies a point of entry to the service that is similar to a traditional interface except that invocation of the command is done in an asynchronous manner and that no results are returned. Outgoing commands specify what in-going commands the service invokes. Both in-going and outgoing commands have types

specified by MIME [17] strings. We adopt the UML lollipop interface notation for commands - provided interface ("closed lollipop") for in-going commands and required interface ("open lollipop") for outgoing commands.

PalCom services can be *bound* or *unbound*. Bound services are tied to the hardware device on which they run, and typically expose functionality in the hardware for remote use. Unbound services, on the other hand, are not tied to the hardware and can thus be moved to and installed on new devices. We use a UML stereotype `<<unbound>>` for unbound services.

3.4 Assemblies

Services are connected by means of *assemblies*. The assembly is PalCom's mechanism for service coordination, central in the project's approach to support for construction and deconstruction of palpable systems. A system that is constructed using assemblies can be inspected in a service browser [18], making its inner structure visible at a certain level. This gives better understanding of the system, and is particularly important when a system breaks down. Furthermore, the assembly concept targets construction of systems from services that were not originally created for cooperation with each other. By inspecting the interfaces of a set of services, it is possible to construct an assembly that combines them. Service composition has previously been used to implement ubiquitous computing applications [19].

Assemblies are defined by assembly scripts that can be loaded at runtime by interacting with an assembly manager (see figure 6). When an assembly is loaded the assembly manager makes the appropriate connections and governs the flow of service invocations. We use the nesting mechanism in UML for assemblies.

One goal of the implementation of the tiles has been to make it possible to replace the game logic easily without rebooting the devices. This is done using assemblies. A set of basic services encapsulates the basic hardware functionality of the tiles and each game is implemented as one or more unbound services that can be connected to these services. The basic services of the tiles are a `LED` service controlling the LEDs, a `Connectivity` service detecting the presence of neighbour tiles, and a `Touch` service receiving input from the button if one is present (as is the case for the assembler tile in the puzzle game). The combination of the assembly and the unbound services for the game logic can be replaced when switching to another game. At present only the puzzle game has been implemented.

The split in functionality between an assembly and an unbound service is normal for a PalCom system. The assembly captures the coordination logic between services, while the services perform most of the calculations. For adding behaviour to a set of services without programming a new service it is possible to express some calculation in assembly scripts, but the assembly script language is intended to be much simpler than the general-purpose programming languages normally used for implementing services. Therefore, complex calculations are implemented in unbound services that are incorporated when constructing an

assembly. Finding the right level of sophistication in the assembly script language, and how much should be delegated to unbound services is a challenge that is under active research in the project.

3.5 The Puzzle Game

As described in [20] each of the tiles can be in one of three states: *sideHappy*, *localHappy*, or *globalHappy*. The states correspond to the types of feedback given by the tiles in the game. In figure 5, the top-right tile is sideHappy in figure 5a, localHappy in figure 5b, and globalHappy in figure 5c. Three rules determine which state a tile is in:

1. A tile is sideHappy if it has less than four correct sides.[5]
2. It is localHappy if it has four correct sides but at least one of the other tiles are sideHappy. This means that the tile has found its place in the puzzle, but the complete puzzle is not solved.
3. If no tile is sideHappy then all tiles are globalHappy. The puzzle is solved.

As can be seen from these simple rules, the game has a notion of global state, namely, whether there is at least one sideHappy node. This information is used by the tiles to distinguish whether the tile is localHappy or globalHappy. If a tile has less than four correct sides it does not need this information (because of rule 1).

The global state is maintained by handling two situations: The first situation occurs when a tile observes that it has four correct sides instead of three. It then broadcasts (by using the publish-subscribe mechanism of the communication model) a challenge to the other nodes requiring any sideHappy nodes to reply immediately (also with a broadcast). If no responses are received within a certain timeframe it is concluded that there are no sideHappy nodes and that the node therefore instead of being sideHappy should be globalHappy. If a response is received the node should be localHappy. When a localHappy node receives a challenge it treats it as if it was originating from itself - the node sets a timer and waits for responses. It is assumed that the solution of the puzzle is connected and includes all nodes and therefore it cannot be the case that no nodes are sideHappy in a proper subset of all the nodes. Therefore, if there is a sideHappy node there is a path from that node to the node that initiated the challenge.

The second situation, inverse to the first one, occurs when a node observes that it has three correct sides instead of four. The new state of the node is now sideHappy. If the node was globalHappy before the other nodes are unaware that the node is now sideHappy, and therefore a message is broadcasted specifying so. If the node was localHappy before, it can assume that there is at least one sideHappy node in the graph it is connected to. The above paragraphs describe how the global state is maintained. Alternately, this could be done using the

[5] We define a correct side of a tile to be a side that has a correct neighbour or has no neighbour and should have no neighbour according to the solution.

two-phase commit (2PC) protocol. The 2PC protocol uses, however, a lot more communication and since the inter-node communication bandwidth is approximately 600 bits/second the protocol has been deemed inappropriate.

Figure 8 shows an UML deployment diagram outlining the structure of the implementation. In the puzzle game there are two types of tiles - the normal tiles and the assembler tile. The normal tiles communicate with each other and with the assembler tile using IR communication. In the normal tiles the `PuzzleAsm` assembly (listed in figure 9) connects the basic services to the unbound services handling the game logic. The `Puzzle` service receives connectivity events from the `Connectivity` service (line 18–20 in figure 9) and determines the local state of the tile. This information is sent to the `LED` service (line 21–26) and to the `Coord` service (line 27–32) that coordinates the global state using the algorithm specified above. If all tiles are correctly aligned the `Coord` service notifies the `LED` service (line 33–35). The assembler tile contains a `Configure` service with the responsibility of initiating and configuring the game and the `PuzzleConfAsm` assembly to connect it to the basic services.

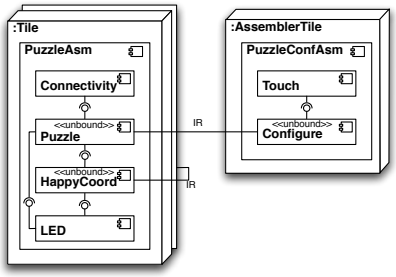

Fig. 8: Services in the puzzle game

```
1 assembly PuzzleAsm {
2   devices {
3     device = urn:palcom://Tile_1;
4   }
5   services {
6     Connectivity on device = Connectivity;
7     Puzzle on device = Puzzle;
8     HappyCoord on device = HappyCoord;
9     LED on device = LED;
10  }
11  connections {
12    Connectivity -> this;
13    Puzzle -> this;
14    HappyCoord -> this;
15    LED -> this;
16  }
17  script {
18    when connectivityUpdate from Connectivity {
19      send connectivityUpdated(thisevent.param) to Puzzle;
20    }
21    when localHappy from Puzzle {
22      send setled('1 1 1 1') to LED;
23    }
24    when sideHappy from Puzzle {
25       send setled(thisevent.sides) to LED;
26    }
27    when localHappy from Puzzle {
28      send localHappy() to HappyCoord;
29    }
30    when sideHappy from Puzzle {
31       send sideHappy() to HappyCoord;
32    }
33    when globalHappy from HappyCoord {
34      send setled('2 2 2 2') to LED;
35    }
36  }
37 }
```

Fig. 9: Puzzle assembly script

4 Evaluation

We argue that a system can be designed to have some palpable behaviour from an activity perspective in the sense that the system is easily perceivable and allows for spontaneous interaction. During normal use it can be very hard for a user to perceive the difference between a system running a palpable framework or not. But if a breakdown occurs, these systems lose all their palpable qualities if the qualities only were implemented 'in the interface'. On the other hand, a system can run the palpable framework, without a user perceiving any palpability in the use of the system. If the system is not designed to communicate its palpability to the users, palpability will not be perceived by the users. But finally, if combined, a much higher level of palpability can be reached within a system. Palpability is not only about internal structure of the software, it is also about communication and interaction.

In Active Surfaces, as in other distributed systems that are characterised by a high level of configurability and limited output capability, the contingencies that might occur over time is of special interest and have to be dealt with. Especially those that can occur in a multi-user environment. Multi-user not only with respect to the number of users, but also with respect to different kinds of users. The challenge here is to allow these users to be in control [16], a key challenge in ubiquitous computing addressed by Palpable computing. We will try to visualise this point with a simple scenario.

> *One day two therapists work at the swimming pool. During the day different children arrive to start their sessions. One of the therapists uses the assembler tile to program and then configure 5 tiles to take part in the therapeutic activity concerning one of the children. She makes a puzzle game with stable and blinking light feedback to indicate the different states of the system. While she is at lunch, her colleague takes 7 tiles to use with another child. By mistake, she includes one of the tiles configured in the 'first' game. As the first therapist returns after lunch, she tries to continue the activity. Now one of the tiles acts in a strange way, or not at all. As the second therapist now has finished her work, the first therapist cannot consult her to realise that she might have altered the game. As the first therapist perceives the situation, the Active Surfaces worked before lunch, and now they do not.*

In distributed and resource constrained systems, many error situations can occur and normally it can be hard for a user to find the reason behind a problem or a mismatch. The assembler tile introduced before in this paper, can, besides being used in the therapeutic activity, also be used as an inspection tool. The therapist can utilise the IR communication protocol to inspect the running services within each tile. Through the inspection the therapist can understand what services and assemblies are running, how they are configured and detect whether there are resource problems that have to be solved. The therapist starts to inspect the game service, and realises immediately that the tile configuration

has been altered. It was not a system error. The therapist reconfigures the tile and can carry on the activity in the swimming pool.

The Active Surfaces system has been developed together with the users (mainly therapists) of the system. The use of the Participatory Design [21] method including different iterations of mock-ups, prototypes and Wizard of Oz [22] sessions indicate that end-users can perceive and control the Active Surfaces as described above. Further full-scale trials have to be carried out to fully support this claim.

The simple scenario demonstrates the need for inspection and user control. Here the therapist initially perceives the behavioural mismatch as a bug or error in the system. In reality all components behave as they should, but one of the tiles have been reconfigured without the knowledge of the current user. It is an example of an error occurring over time in one of the distributed components, even when the system should have been idle. The user must be provided with tools that allow him to understand or 'look inside' the system to overcome the mismatch. It is important that the user understands that this is not an error; it is a misconfiguration that can be overcome.

5 Conclusions and Future Work

In this paper we have described the implementation of the Active Surfaces prototype, which is used in physical and cognitive rehabilitation work. We have shown in an example scenario how palpable qualities in a system can be valuable when the system behaves in unexpected ways. In those situations, it is beneficial for the user to have a system that is built as a set of services combined into assemblies. Palpable system allows for inspection, so errors and misconfigurations can be located and corrected by end users.

The Active Surfaces prototype has been implemented in a distributed, embedded system, executing in a set of floating tiles, and in a simulation framework running on a standard PC. Experiences gained during the work on the prototype provide input to the on-going development of the PalCom assembly concept. The prototype implementation has helped concretise requirements for supporting more powerful coordination logic, including coordination based on broadcast communication. The structure of the tile games calls for assemblies that span over multiple physical devices, and for decentralised assemblies that do not require particular devices always to be present. When assembly descriptions can express such behaviour, less coordination logic has to be delegated to unbound services.

Acknowledgements Thanks to Laura Cardosi, parents and children for their open-minded collaboration and continuous support during fieldwork and participatory design. We also would like to thank our colleagues at our universities, especially Alessandro Pollini, Patrizia Marti, Alessia Rullo, Boris Magnusson, Jacob Frølund, Henrik Gammelmark, and Mie Schou Olsen. The research was part of the European IST project PalCom.

References

1. Weiser, M.: The computer for the 21st century. SIGMOBILE Mob. Comput. Commun. Rev. **3**(3) (1999) 3–11
2. Buxton, W.: Integrating the periphery and context: A new model of telematics. In: Proceedings of Graphics Interface. (1995) 239–246
3. Weiser, M., Brown, J.S.: Designing calm technology. PowerGrid Journal (1996)
4. PalCom - making computing palpable: http://www.ist-palcom.org.
5. Grönvall, E., Marti, P., Pollini, A., Rullo, A.: Active surfaces: a novel concept for end-user composition. In: NordiCHI '06: Proceedings of the 4th Nordic conference on Human-computer interaction, New York, NY, USA, ACM Press (2006) 96–104
6. PalCom: PalCom External Report no 50: Deliverable 39 (2.2.2): PalCom Open Architecture. Technical report, PalCom Project IST-002057 (2007)
7. Svensson, D.: PalCom Working Note #112: Service framework. Technical report, PalCom Project IST-002057 (2006)
8. Marti, P., Rizzo, A.: Levels of design: from usability to experience. In: Proceedings of HCI International. (July 2003)
9. Marti, P., Moderini, C.: Creative design in safety critical systems. In: Proceedings of ECCE 11. (September 2002)
10. Corry, A.V., Hansen, K.M., Svensson, D.: Traveling architects – a new way of herding cats. In: Quality of Software Architectures. (2006) 111–126
11. Büscher, M., Kristensen, M., Mogensen, P.: Making the future palpable: Notes from a major incident future laboratory. In: Proceedings of the 4th International Conference on Information Systems for Crisis Response and Management (ISCRAM). (May 2007)
12. uClinux: http://www.uclinux.org/.
13. Myers, B.A.: Visual programming, programming by example, and program visualization: a taxonomy. SIGCHI Bull. **17**(4) (1986) 59–66
14. Montemayor, J., Druin, A., Farber, A., Simms, S., Churaman, W., D'Amour, A.: Physical programming: designing tools for children to create physical interactive environments. In: CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems, New York, NY, USA, ACM Press (2002) 299–306
15. UNC20: http://www.unc20.net/.
16. Schultz, U.P., Corry, E., Lund, K.V.: Virtual machines for ambient computing: Virtual machines for ambient computing: A palpable computing perspective. Presented at Object Technology for Ambient Intelligence Workshop, ECOOP (2005)
17. MIME Media Types: http://www.iana.org/assignments/media-types/.
18. PalCom: PalCom External Report no 57: Deliverable 43 (2.6.2): End-User Composition: Software support for assemblies. Technical report, PalCom Project IST-002057 (2007)
19. Brønsted, J., Hansen, K.M., Ingstrup, M.: A survey of service composition mechanisms in ubiquitous computing. To appear in 'Second Workshop on Requirements and Solutions for Pervasive Software Infrastructures (RSPSI)' at Ubicomp 2007
20. Grönvall, E., Pollini, A., Rullo, A., Svensson, D.: Designing game logics for dynamic active surfaces. Presented at MUIA 2006: third international workshop on mobile and ubiquitous information access (2006)
21. Greenbaum, J., Kyng, M.: Design at work: cooperative design of computer systems. Lawrence Erlbaum Associates, Inc. Mahwah, NJ, USA (1992)
22. Erdmann, R.L., Neal, A.S.: Laboratory vs. field experimentation in human factors–an evaluation of an experimental self-service airline ticket vendor. Human Factors **13**(6) (1971) 521–531