An Open Architecture for Palpable Computing

Some Thoughts on Object Technology, Palpable Computing, and Architectures for Ambient Computing

Peter Andersen, Jakob Eyvind Bardram, Henrik Bærbak Christensen
Aino Vonge Corry, Dominic Greenwood, Klaus Marius Hansen, Reiner Schmid

The PalCom Project, `http://www.ist-palcom.org`

Abstract. Ambient intelligence and ambient computing is concerned with providing people with information, means of communication, and entertainment at any time and at any place. This vision is implemented by distributed populations of devices capable of adapting their characteristics according to the extant and anticipated needs of users. *Palpable Computing* (PalCom) complements and extends the tenets of ambient computing by focusing on the user experience through control, understandability, de-construction of systems and variable visibility of underlying computational process. This paper first presents an Open Architecture proposed to support the challenges and requirements of palpable computing infrastructures. It then moves on to discuss the implications of PalCom to object technology.

1 Introduction

Consider the following scenario:

A group of paramedics arrive at a large accident scene with more than 20 injured people. They start attaching medical sensors to each patient while ensuring that data is relayed via ambulances back to the hospitals' emergency centres. The software services on each sensor work together within the assembly of the set of sensors, attached to a particular patient. The task of sensing data, coordinating the sensors to a specific patient, identifying this patient, and to relay this data away from the patient is handled by a sensor assembly software component. A hand-held device used by the paramedics can discover and query a patient's sensor network when within range. When sensors experience network problems – e.g., if sensors on the front and back of a patient cannot see each other because of the water in the human body – they adapt their behavior to accommodate the changing network coverage. Seen from the emergency centre at the hospital, the scope of the accident is the components, sensor assemblies, services, and actors involved in the accident. They may be handling other accidents with a separate scope. Moreover, the surgeon on duty on the emergency centre allows the paramedics at the site to access the hospitals medical record service. While paramedics dispatch patients to ambulances which take them to different hospitals, the paramedics on site are able to maintain contact to the patient, i.e., to the computational assembly surrounding the patient even when the patient is changing context from site to ambulance to hospital, etc. And, when the patient is moved to the ambulance, some of the sensors are removed and reused on other patients.

Seen from a software engineering perspective, the above scenario could be realized using a range of existing ubiquitous and ambient computing technology (with a great deal of effort, however). Seen from a software architecture research perspective, the above scenario poses a range of questions which is primarily related to how the users would actually experience and manage a complex computational setup with the hundreds of wireless computers in a small space, collaborating with background, infrastructure services. These questions include:

- While supporting an exceedingly *heterogeneous* and *ever changing* computational environment, such as the accident scene, how can we help users manage their tasks, whilst maintaining *coherence* and *stability* in the supporting systems without the need to focus on directly handling heterogeneity. For example, how can we support the paramedics and the clinicians at the hospital to focus on data from one patient while the sensors and the context is changing? The system should quietly support human tasks; not be intrusive, obtrusive or interfere unless expected or told to do so.
- How can we support the large number of devices in this scenario while still ensuring that users understand what is happening in critical situations such as this one? How can the user investigate and render *visible* the inner workings of the technology in order to *understand* what is going on? For example, if data from one patient indicates a serious event, how could the paramedics identify quickly if it is genuine or result of a sensor malfunction?
- How can users use and re-use software components, services, and devices without an enormous configuration overhead? How can we help users to *reconstruct* new configurations from existing ones? For example, how can sensors in the accident be re-used and how can on-site medical data be enriched by medical data from the hospital? And how can users *control* and manipulate this constant reconstruction and resource negotiation.
- How can we handle *errors* and *failures* in a highly complex and heterogeneous multi-user, multi-device, multi-service environment? How can we ensure that contingent mechanisms are in place to automatically handle problems while simultaneously introspecting the system to inform the user of the problem origin. This must be managed through close cooperation between the user and system whilst avoiding the necessity for close coupling which might cause problem conditions to permeate and cause wider errors or failures.

These challenges and others are addressed in the EU-funded Integrated Project PalCom [13]. This project uses the term 'palpable computing' to denote a new kind of ambient computing which is concerned with the above user-oriented challenges in complex and dynamic ambient computing environments. One of the primary goals of the PalCom project is to design an open software architecture for PalCom. This paper presents the design of this architecture and discusses how object-oriented concepts and technologies play a core role in it.

2 Architecture Overview

The Open Architecture is the technical nucleus of the PalCom project. Its goal is to serve as the means of transcribing the challenges discussed in the previous section into a set of interrelated, computationally realised concepts drawn into a coherent, encompassing and meaningful architecture. This implies that it must capture the essence of how human actors interact with, and within, their everyday environments through distributed populations of palpable and non-palpable entities.

The architecture builds upon established concepts in relevant fields of software engineering expertise, especially object-oriented design. But it also specialises this know-how to weave a computational fabric that, as the sum of its parts, delivers the means to pragmatically enable aspects of palpability in real and useful ways.

The PalCom open architecture is a service-oriented architecture which forms the basis for creating *PalCom Systems* which primarily consist of a set of networked *Devices* hosting a population of *Services*. A coordinated group of devices is known as an *Assembly* that can be regarded and reasoned about as a whole by either the user or some other computational entity capable of reasoning (such as the sensor assembly example in the introduction) [9]. A Service is defined as a remotely accessible, discoverable, self-contained runtime component that can be composed into aggregates accessible through a single exposed interface. Each Service consists of one or more Runtime Components which are in turn deployed instances of PalCom Components - the fundamental level of computational encapsulation provided by the architecture.

A PalCom System typically consists of services and assemblies running on multiple networked devices that together populate a user's environment.

As such, services and assemblies provide the top layer from a user and developer perspective. Figure 1 shows this layer and the remaining layers of the PalCom architecture: The bottom layers represents the *Hardware* and the optional *Operating System* of devices participating in PalCom systems. These two layers are considered as pre-requisite and are therefore not subject to any special treatment with the architectural scope. The next layer, the *Runtime Environment* is the lowest aspect of the PalCom Architecture and is responsible for hosting PalCom Services on devices. With a PalCom Runtime Environment in place a device can then be considered as being a PalCom-enabled device. The *Common Infrastructure* provides operational support for PalCom Services and Assemblies, including resource management, contingency management and security.

Beyond specifying the nature of these layers, the Open Architecture is supported by a *programming model* for designing PalCom Components and for orchestrating PalCom Services. The run-time environment and the common infrastructure layers are described in more detail in Section 2.1 and Section 2.2 below.

The architecture is in the process of being validated by a number of *architectural prototypes* [5] as work streams within the PalCom project.

2.1 Runtime Environment

The PalCom runtime environment acts both as a platform and as a resource to PalCom entities. It is therefore necessary that at least one instance of the PalCom runtime en-

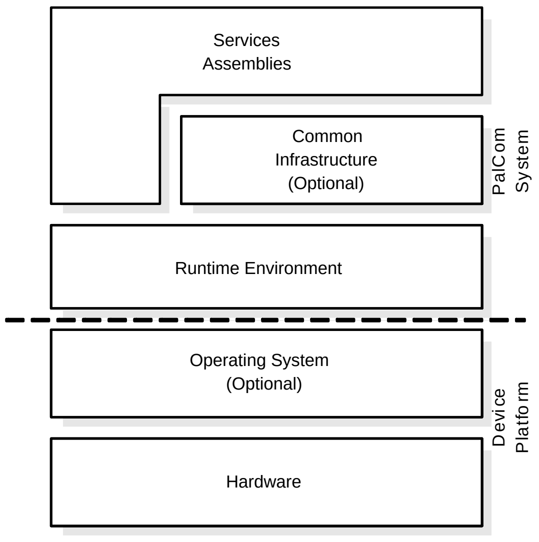

Fig. 1. Logical view of the PalCom architecture layers

vironment is located at and available on each PalCom Device. In addition, the runtime environment is the only element of the PalCom Open Architecture that explicitly relies on the existence of a network and the capability of PalCom Devices to join and leave this network when needed.

The description of the runtime environment is divided into two interrelated aspects. The *component model* defines the nature and deployment of PalCom Components, as well as the creation of PalCom Services and PalCom Assemblies. The *communication model* defines the mechanisms necessary to provide PalCom Components with remote communication endpoints, thus transitioning them into PalCom Services. The component and communication models as well as the optional virtual machine of the PalCom architecture are briefly outlined next.

Component Model. A PalCom *Component* is a unit of functionality, deployment, and composition with specified interfaces and dependencies to other components [14]. A PalCom Component consists of three parts: interfaces, an implementation, and annotations. One or more *interfaces* define the contract of the component, such as the published functionality and quality of service. An *implementation* realizes the contracts specified in the component's interfaces. *Annotations* provide information necessary to use and execute the component appropriately, for instance, versioning information and resource requirements for devices on which it should run.

A PalCom *Service* is deployed as a PalCom Component and instantiated as a PalCom Run-time Component. When instantiated it encapsulates a remotely accessible, discoverable, self-contained, context-independent, and reactive element of domain or infrastructure functionality. Examples of services from the accident scenario are a medical sensor data service, a global positioning service, and a persistence service. Preferably, services are implemented as stateless, in the sense that they do not remember invoker state from one invocation to the next. This is done in order to minimize impact on the performance and scalability of PalCom systems.

A PalCom *Assembly* is a mechanism that allows individual PalCom enabled devices to be coordinated via composition and orchestration of the services they host. A PalCom *Assembly* can be deployed as a *composition and coordination specification*. Composition is the ability to specify and scope a set of PalCom Services that contribute toward the behaviour of a PalCom Assembly. Coordination describes the interaction of services within the context of an assembly of PalCom enabled devices.

Communication Model. PalCom devices are required to be network enabled to allow services to communicate. Two central aspects of the communication model are to establish a connection and to use the connection for control and data exchange.

To support this the PalCom architecture defines a language-independent *announcement and discovery protocol* that enables mutual awareness and detection of PalCom services, assemblies, and the devices hosting them. A PalCom Service will, upon instantiation, make an announcement which is disseminated by the runtime environment to all receiving devices on the network to which the device is connected. Each announcement contains a language-independent service descriptor that describes the functionality provided, the interfaces, and the communication protocol to use. Received announcements are then forwarded, according to any constraining authorisations, to any hosted Service that has registered an interest in (via subscription) the particular discovery announcement. As such the protocol has no centralized entities.

Having detected each other, services are then able to make use of one anothers functionality. Typically, services have no explicit object references to each other, but rather collaborate by communicating using open interfaces that are remotely accessible. The communication protocol supported is either *message-oriented* or *proxy-based* for which the infrastructure defines an abstraction layer encapsulating the actual network technology used by the device.

The message-passing paradigm allows for time, space and flow-decoupling between the communicating services. Using this scheme, services can communicate in a group-oriented manner by subscribing to some pattern that messages published by other services are then matched against.

Services can also provide proxy objects for proxy-based communication. This ensures synchrony between services and grants advantages such as type safety, protocol encapsulation and call-site invocation metadata. The run-time environment handles transferring the proxy object to the service requester site. As far as concrete network protocols are concerned, proxy objects can both leverage protocols from the communication infrastructure and encapsulate a private protocol to contact their service. The latter option could, for example, allow the provision of secure communication via a specified encryption mechanism.

Virtual Machine. To support portability across multiple hardware devices and operating systems it is recommended to use a virtual machine part of the runtime environment.

For reasons of, e.g., legacy code or device resource constraints this may not always be possible; an example would be some of the medical sensors from the emergency scenario above. The specification does however dictate, that if a virtual machine is used, it must comply to the PalCom Virtual Machine.

The PalCom Virtual Machine is specified as a very simple, light-weight virtual machine with a minimal memory footprint while still providing among others portability and mobility support. The virtual machine is specified to be able to run directly on top of a given microprocessor without requiring an underlying operating system, though running on top of most operating systems is also possible (akin to the virtual machine of [1]).

The virtual machine is also designed such that it can be implemented and optimized to run on devices with memory requirements in the range of 128K bytes. The virtual machine has a common object format and common binary component format and is designed to be language independent. The virtual machine supports automatic memory management including garbage collection.

Finally, the virtual machine contains a remote service interface which supports remote observation, testing and modification of components running on a given device, and it supports error-handling, communication, reflection and introspection.

2.2 Common Infrastructure

The *Common Infrastructure* resides on top of the runtime environment and provides a set of common infrastructure services: *Resource Management*, *Contingency Management* and *Security* which each add a layer of *quality of service* to PalCom systems through the features they provide for managing and manipulating PalCom components, services and assemblies. These services are viewed as being of significant value when implementing a PalCom system, but are not considered mandatory in order to differentiate between baseline features, such as communication, and secondary infrastructure features that whilst important for creating resilient, well managed systems, are not necessary for the basic implementations.

Resource Management is the means by which PalCom Devices, Components, Services, and Assemblies in a PalCom system are aware of and can manipulate resources available for consumption within their operational environment, including themselves. The PalCom notion of a resource is drawn from the definition made by Kircher and Jain [10], who defines a resource as “an entity that is available in limited supply such that there exists a requester, the resource user, that needs the entity to perform a function, and there exists a mechanism, the resource provider, that provides the entity on request”. A PalCom resource can be classified into three types: physical, logical and functional. *Physical resources* tend to be characterised as being tangible and include human users, devices, memory, files, network media, processor cycles, power supply, etc. *Logical resources* are the computational abstractions placed over physical resources in order to manipulate them, such as threads, network connections and file handles. *Functional resources* are resources that have a programmatic nature, such as software components, services and assemblies.

The PalCom runtime environment manages the physical resources of the host device, exposing their availability and usage characteristics, as logical resources, to the common infrastructure layered over the runtime environment. The *Resource Management* service of the common infrastructure is responsible for manipulating logical and functional resources. This is an optional service whose functionality can vary according to the particular device, but will typically include; discovery, registration, reservation, allocation, distribution, pooling, re-configuration, and migration of resources.

By way of example, it may often be the case that a service requiring a particular resource cannot be started due to the absence of that resource on the local device. The resource manager service can then be used to identify alternative resources, perhaps a remote device, some distributed shared memory or a secondary service with a similar functional semantics, to substitute for the absent primary.

Contingency Management ensures that in the event of an erroneous or inappropriate condition, actions are available to adapt system behaviour to compensate in the most appropriate way. This might imply switching to an alternative network access point or service, using memory on another device, displaying information via alternative means, etc. A key issue is ensuring correct dependency tracing to guarantee that any and all affected elements of a PalCom system are either notified or automatically adapted to compensate for changes induced by error conditions.

In compliance with the idea of PalCom, such contingency management efforts may often involve the user by ensuring that they are made aware of events and given the option to intercede in decision making that might otherwise be deferred to automated adaptation.

The design for contingency management is inspired by Erlang [2,3]. Contingency management in the PalCom architecture has three essential requirements:

- Detection – It must be possible to detect and handle problem conditions both *locally* (regular exception handling in the component) as well as *remotely* (in non-local components).
- Isolation – Problems occurring in one component must be prevented from cascading and thereby causing problems in other parts of the the system.
- Treatment – It must be possible to determine why a problem condition has occurred and apply preventative and/or remedial actions to treat it.

The communicative nature of service interaction in PalCom is fundamental to ensure these requirements. In a PalCom system, we expect all services to be inherently unstable and unreliable. Therefore, one service cannot rely on another, and errors in one service should hence not fundamentally affect other services. For example, in the emergency scenario when the sensor starts to malfunction, this must not affect the behaviour of the assembly. The assembly keep on working even though it no longer receives information from this sensor, implying that appropriate contingent actions must be taken to work around the problem and ensure uninterrupted operation.

Specific *Contingency Management Handlers* can specify different behaviour in different contingency situations. Due to the strong binding between contingency management and resource management, many actions taken to resolve fault conditions will typically be taken at the level of the local resource manager of a PalCom device.

Currently, the PalCom architecture differentiate between *exceptions* and *errors*. Exceptions are defined as non-critical events that are expected to be resolved at the level of the service or assembly which raised them. Errors however, are defined as fault conditions that a service or assembly is unable to deal with itself and thus must be managed at the level of the Resource Manager or other suitable infrastructure service.

3 Object Technology and PalCom Architecture

Based on our experience, object technology can bring both benefits and liabilities to both ambient and palpable computing.

3.1 Visibility and composition

One of the main challenges of the PalCom project has been to reach the best tradeoff between *visibility* and *invisibility*. In order to obtain invisibility we must hide some details according to changing context. This is an inherent benefit of object technology by means of *encapsulation*. But, we also need to expose parts of the inner structure in order to effectively manage introspection for contingency management. This may in part be obtained by use of interfaces to the objects/components. It will not reveal the actual inner structure, but as much of it as the designer has predicted the need for.

To support *composition/decomposition* the encapsulation of state and interfaces to the environment is also a great benefit. Again the designer will decide on what level the environments can be composed and decomposed. But likewise, the granularity of the compositions are restricted by the expectations of the designer.

3.2 Hierarchical Maps

As a research issue we have explored the pros and cons of encapsulation of the objects in the settings. An experimental framework has been developed, Corundum [15]. This framework facilitate a high degree of remote introspection and visibility of the state of the internal nodes: All of the Corundum framework is based on a *hierarchical map* (h-map) concept which provides an “exoskeleton”[1] to the process. The h-map is the structure which holds together the different software parts of the process, and is externally visible. The h-map is a per-process hierarchical map. Inspired by the Plan 9 file system concept and the `/proc` file system of Linux we use the h-map to map names to entry points, objects, meta data, descriptions of interfaces, etc. Basically, all non-transient data is kept in the h-maps. The h-map thus enables introspection and remote manipulation of all nodes, aiding the visibility aspect of Palcom. This clearly breaks with the OO-concept of data encapsulation, but compared with the traditional OO approach of using reflection for this, we get a much more open and uniform access to internal features of the process. One may argue that this gives a too open access to the features of the process, and to circumvent this a permission facility has been added, which supports various levels of access.

[1] See e.g., `http://en.wikipedia.org/wiki/Exoskeleton`

3.3 Detached Reflection Server

As an alternative to the h-map approach we have also investigated modifying traditional reflection models to better suit the needs of PalCom. The main issue being to support introspection, even on very small devices. In [6] it is argued that reflection capabilities should be kept separate from the base class level. On very small devices the overhead of deploying the meta-data needed for traditional reflection may be too large. By extending the idea of the detached reflection mirrors, we have experimented with an idea of keeping the meta-data completely separated from the code on a dedicated device, a "meta-data server", thus preventing replication. One could stretch the point and say that h-maps are actually a form of mirrors, at least as seen by the user of services described in the h-map.

3.4 Dynamically Typed VM

The need for supporting, e.g. change and reconstruction, has made us experiment with a virtual machine design based on a *dynamic type system* in the tradition of SmallTalk[8], instead of a static type system, like, e.g. JVM[11] and .Net[7]. This does not mean that we enforce programming in SmallTalk: Bytecode compilers have been implemented for the statically typed languages Java[4] and BETA[12]. Our experiments show that supporting a statically typed language on a dynamically typed VM is more straightforward than the opposite.

4 Conclusion

In the work conducted to date we have defined an architectural basis for palpable systems. The ambiguity, complexity and special needs of these systems forces us to explore extreme applications of object technology. In our research we have made a number of design decisions and discovered many open issues. This paper has touched upon many of them which we hope will lead to a fruitful discussion of the various technologies, designs and implementations that support and impact palpable systems.

Acknowledgements

The research presented in this paper has been partly funded by 6th Framework Programme, Information Society Technologies, Disappearing Computer II, project 002057 'PalCom: Palpable Computing – A new perspective on Ambient Computing' (`http://www.ist-palcom.org`)

References

1. Jakob R. Andersen, Lars Bak, Steffen Grarup, Kasper V. Lund, Toke Eskildsen, Klaus M. Hansen, and Mads Torgersen. Design, Implementation, and Evaluation of the Resilient Smalltalk Embedded Platform. *Computer Languages, Systems & Structures*, 31(3-4):127–141, 2005.

2. J. Armstrong, R. Virding, C. Wikstrom, and M. Williams. *Concurrent Programming in Erlang*. Prentice-Hall, 1996.
3. Joe Armstrong. *Making reliable distributed systems in the presence of software errors*. PhD thesis, Royal Institute of Technology, Swedish Institute of Computer Science (SICS), Stockholm, December 2003.
4. Ken Arnold, James Gosling, and David Holmes. *The Java Programming Language*. Addison Wesley, third edition, 2000. `http://java.sun.com/docs/books/javaprog/`.
5. J. Bardram, H. B. Christensen, and K. M. Hansen. Architectural Prototyping: An Approach for Grounding Architectural Design and Learning. In *Proceedings of the 4th Working IEEE/IFIP Conference on Software Architecture (WICSA 2004)*, pages 15–24, Oslo, Norway, 2004. `http://csdl.computer.org/dl/proceedings/wicsa/2004/2172/00/21720015.pdf`.
6. Gilad Bracha and David Ungar. Mirrors: Design principles for meta-level facilities of object-oriented programming languages. In *Proceedings OOPSLA 2004*, volume 39(11) of *ACM SIGPLAN Notices*, pages 345–364, Vancouver, BC, October 2004. ACM.
7. Standard ECMA-335. *Common Language Infrastructure (CLI) Partitions I to V*. ECMA International, 2nd edition, December 2002. `http://www.ecma-international.org/publications/files/ecma-st/ECMA-335.pdf`.
8. Adele Goldberg and David Robson. *Smalltalk-80: The language and its Implementation*. Addison Wesley, 1983.
9. Mads Ingstrup and Klaus Marius Hansen. Palpable Assemblies: Dynamic Service Composition in Ubiquitous Computing. To appear in Proceedings of Software Engineering and Knowledge Engingeering (SEKE) 2005, 2005.
10. M. Kircher and P. Jain. *Pattern-Oriented Software Architecture: Patterns for Resource Management*. Wiley, 2004.
11. Tim Lindholm and Frank Yellin. *The Java(TM) Virtual Machine Specifi cation*. Addison-Wesley, 2nd edition, 1999. `http://java.sun.com/docs/books/vmspec/`.
12. Ole Lehrmann Madsen, Birger Møller-Pedersen, and Kristen Nygaard. *Object-Oriented Programming in the BETA Language*. ACM Press/Addison Wesley, 1993.
13. 6th Framework Programme, Information Society Technologies, Disappearing Computer II, project 002057 'PalCom: Palpable Computing – A new perspective on Ambient Computing'. http://www.ist-palcom.org.
14. C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 1998.
15. Peter Ørbæk. Programming with hierarchical maps. Technical Report DAIMI PB-575, DAIMI, 2005. `http://www.daimu.au.dk/publications/PB/575/PB-575.pdf`.