# Middleware and Application Adaptation Requirements and their Support in Pervasive Computing

Christian Becker and Gregor Schiele
University of Stuttgart
Institute for Parallel and Distributed Systems (IPVS)
Universitätsstr. 38, 70569 Stuttgart, Germany
(christian.becker / gregor.schiele)@informatik.uni-stuttgart.de

## Abstract

*Pervasive computing environments are characterized by an additional heterogeneity compared to existing computing infrastructures. Devices ranging from small embedded systems to full-fledged computers are connected via spontaneously formed networks. In this paper we analyze requirements of applications and system software to cope with the dynamically changing execution environment. Based on our micro-broker-based middleware BASE a component framework for pervasive computing supporting application adaptation is proposed.*

## 1. Introduction

In the recent past, middleware platforms have been the target of researchers in order to provide flexibility with respect to the configuration of the middleware itself. Requirements on such reconfigurable middleware systems arose mainly from the domain of Quality of Service (QoS) management. Different application requirements on non-functional aspects, such as QoS, lead to mechanisms of the middleware to ensure a distinct QoS property.

The vision of ubiquitous or pervasive computing adds new complexity. Our everyday environment becomes populated with smart 'everyday items'. That is, processors are integrated into the environment and allow to access information related to the real-world as well as to control distinct functionality. The end systems in such scenarios are far more heterogeneous than in classical computing environments. Sensors will only need limited computing and communication capabilities and other devices will be dedicated to a single purpose, i.e. a presentation system in a video projector might contain a full-fledged computer but its software is specialized for presentation management. Besides the involved devices, the communication technology will differ as well, ranging from infrared connections over radio links to computers connected via static links. The resulting network topologies will frequently change due to user and device mobility. Information and services available are bound to the location of the device, e.g. temperature information or a presentation system at a far away place are typically less interesting than those available nearby.

As a result, the requirements on adaptation and configuration of the underlying middleware as well as those from the applications change compared to those requirements already present in classical middleware systems. In this paper we will present an example scenario, derive and motivate requirements on middleware configuration support and application adaptation. Our approach to support these requirements via a micro-broker based middleware – BASE – and a component model based on an application framework is then presented. After a discussion of related work the paper closes with a summary and outlook on future work.

## 2. System Model

This section will first present a pervasive computing scenario and two possible applications before the system model is defined.

### 2.1. Scenario

Let us consider a scenario as it is common in the envisioned pervasive computing systems. Present in such a scenario are embedded and specialized devices, e.g. sensors providing information about temperature, position of users or specialized systems, such as the before mentioned presentation system. All these devices are equipped with wireless communication. Along with these stationary devices, mobile devices which are typically carried by users are present. Such devices could be handheld devices, such as personal digital assistants (PDAs) or cell phones, but in the future there might be smart clothes as well. The computing environments of today will not vanish or be substituted by these devices but complement such systems. In order to motivate the use of such environments to applications two possible applications are sketched:

1. Support of senior citizens: in order to support the life of the elderly in their home, their body functions and positions might be captured and evaluated at a designated home server. If a change in the health condition occurs, information how to behave is presented through audio or video devices in the room where the person currently is located. In serious health conditions an ambulance is called and provided with the health status of the person.

2. Office support: the status of rooms and objects could be monitored by sensors and propagated into the vicinity. Users nearby are thus provided with environmental information as well as vacancies of meeting rooms etc. Additionally, locally available services become accessible when a user is nearby, e.g. a presentation system is only of use, when the user is in the same room to make use of its output.

Before we will derive requirements from these scenarios the underlying system model will be presented.

## 2.2. System Model

Pervasive computing environments can be classified by the involved devices and the network characteristics. Furthermore, applications depend on the abstractions provided by the underlying operating system or middleware – which is referred to as system software. We will briefly sketch the characteristics of these three topics in the remainder of this subsection.

**2.2.1. Devices.** As stated above, devices range from sensors over specialized systems to full fledged computers and mobile devices. Besides their processing and storage properties – which may differ widely – devices provide different capabilities which can be used by applications running on these devices. Examples are sensors, e.g. temperature as well as positioning, display or input capabilities, or some controlling capability, such as dimming the light or adjusting the blind of a window.

The availability of a device capability might be restricted in space and time. A GPS sensor is not likely to work within a building and at night sensors based on daylight will stop operating.

**2.2.2. Network.** The wireless connections between the devices differ with respect to the underlying technology and their characteristics. The most profound difference to classical computing environments is the spontaneous nature of such networks, which are formed by nodes which are temporarily in each others communication range. Obstacles, user mobility, and power saving are common events which lead to a reconfiguration of a spontaneous network.

As a result, services located on a device that is not in the current spontaneous network of a client, are not available. This prevents the usage of centralized lookup or trading services. During the interaction with a service, the device providing the service might leave the network. Since devices can be equipped with different network interfaces, spontaneous networks will overlap, i.e. some devices might be reachable via more than one network interface at a time.

**2.2.3. System software.** The support of system software clearly may differ widely. Specialized operating systems for embedded devices, common operating systems with middleware support, or completely proprietary solutions are present. From an application point of view, the abstractions how to interact with remote services – a typical middleware responsibility – and how to access device capabilities, which is an operating system task, are important in order to be comprehensive and yet easy to use.

Another relevant issue in distributed systems in general is interoperability which is typically achieved by relying on interoperability protocols.Interoperability protocols reflect the communication model of the application as it is supported by a middleware. Remote method invocations are reflected by request/response messages while events can be realized by one-way messages containing the event. Requirements on the underlying transport, such as an error-free connection-oriented channel, lead to a restricted usage if only one-way communication – via an optical link, or connection-less communication – is available.

## 3. Requirements

In this section we will derive requirements on the adaptation of applications and its support by system software and component models.

Applications are considered to be executed in a distributed way. Standalone applications could require adaptation support as well, e.g. when a device capability becomes unavailable (a GPS sensor indoor), but these kinds of adaptation requirements are a subset of the more general ones of distributed applications.

## 3.1. Application Adaptation Requirements

Applications in a spontaneous networking environment have to cope with:

**Changing service and device capability availability:** With devices becoming available their services should be used by an application. As well, if a service becomes unavailable, an alternative service should be selected. This will only work, if applications are composed of services with clear dependencies. If alternative levels of an application are defined which require different services the application can continue as long as at least one set of services is available. Clearly, this cannot be supported by the middleware alone but requires an appropriate framework for applications. The same mechanisms can be used to address fluctuating sensor availability on a device.

**Different abstractions for programming of device capabilities:** Middleware and operating system abstractions for remote services and device capabilities are typically different, e.g. proxy objects vs. system calls. This hardens adaptation, since the switch of a local to a remote device capability cannot be done with the same programming interface.

### 3.2. System Software Adaptation Requirements

System software in a spontaneous networking environment has to support:

**Device lookup and service discovery for spontaneous networks:** The device lookup depends on the underlying network characteristics and thus requires distinct lookup mechanisms for each supported network interface. Additionally, services might require distinct interoperability protocols which also depend on the network interface and hence the service lookup will have to take this into account. The detection of lost devices and thus the unavailability of all services in use on that devices have to be signalled to the application or an application framework.

**Flexible protocol support and selection:** If a network interface looses its connection to another device, communication should be upheld if other network interfaces can provide a communication channel. Switching between different interoperability protocols over networks with different characteristics however requires adaptation if the underlying transport does not fulfill requirements of the interoperability protocol, e.g. IIOP requires connection-oriented error-free/signalling communication.

**Decoupling of application communication model and interoperability communication model:** In order to allow different communication links for outgoing and incoming messages, the application communication model, e.g. RPC or events, should be kept independent from the communication model of the possible interoperability protocols. For example this allows communication over infrared via sending out a request as an event and receiving the reply as a reply message over an RPC interoperability protocol based on TCP and IEEE 802.11.

**Uniform abstraction for device capabilities and services:** This allows applications to access remote capabilities in the same way as local ones. Moreover, a uniform abstraction to access services and device capabilities allows to mask the heterogeneity of devices.

**Flexible integration of adaptation mechanisms:** Since different application requirements will need support through mechanisms, e.g. to migrate a component to a remote host to increase the application performance or to migrate it to the local node in order to save energy, different mechanisms should be easily integrated, configured, and used either directly by an application above the system software or by an application framework.

A system software offering the above mentioned support is not sufficient to help application programmers to conquer the heterogeneity and dynamics of pervasive computing environments. Instead of programming towards middleware mechanisms and selecting distinct mechanisms manually, application programmers should rely on high level policies which will result in combinations of mechanisms of the system software. Examples for such policies are ’Energy-Saving’, leading to fostering local execution of application components and restricting radio communication that is costly in terms of energy, or ’Increasing-Availability’, which would make extensive use of remote services in order to allow the application execution – as a trade-off to energy.

What is needed in addition to a middleware supporting the requirements stated above is an application framework that will provide benign abstractions for choosing appropriate adaptation policies. In order to support such a framework, we have developed a micro-broker-based middleware, BASE [2], which meets these requirements. Currently we are developing a component system based on BASE which will allow the specification of component dependencies.

In the following we will sketch the design of BASE and the component system, which we are currently developing.

## 4. BASE a Micro-broker based Middleware

Our middleware BASE is intended to be a minimal platform suitable for small embedded systems but extensible to make use of abstractions available on resource-rich environments. BASE provides application programmers with suitable abstractions to conquer the heterogeneity in pervasive computing environments. Another objective of BASE is to form a foundation for an adaptation supporting component framework. We will briefly sketch the overall architecture of BASE. More detailed information is available in [2].

The major design decision in BASE was to choose a micro-broker design. Device capabilities as well as local and remote services are uniformly accessible via invocation objects, which carry the target object, method-name, parameters, and a service context indicating special handling of the invocation, such as QoS parameters. The micro-broker takes incoming invocations and dispatches them to either a local service via a skeleton, a remote service via a transport module which connects the local and remote device, or to a device-local device capability. Hence, remote device-capabilities can be accessed as services as well.

Invocation objects can be created manually or – if a service provides a stub object – through a proxy (stub object) as conventional middleware systems typically provide. The micro-broker is responsible for synchronizing the caller and issue invocations and receive possible replies as well as an invocation. This allows the application to choose different communication models, such as remote-procedure-calls (RPC), deferred synchronous RPCs, or events via stub objects. Furthermore, the utilization of different interoperability protocols becomes possible. Interoperability protocols typically reflect the applications communication model. However, since the micro-broker maps the application communication model to an exchange of invocation objects, different protocols can be used as long as they accept an invocation object and transfer it. Using the same interoperability protocol for out-going and incoming invocations is not necessary, since the micro-broker keeps track of expected responses (modelled as invocations as well). A scenario where a node uses two communication technologies for the out-going request and the incoming reply is depicted in Figure 1.

BASE allows the integration of transport plug-ins during runtime. The dynamic invocation creation along with local service registries provide a simple reflection mechanism.

The BASE prototype has been implemented in Java, making it suitable for a variety of Java-enabled embedded systems, e.g. mobile phones or the TINI-Board. A minimal configuration of BASE requires 130 KBytes of memory. Due to buffer usage this can increase to a maximum of about 400 KBytes. Still, this makes BASE suitable for many small embedded Java-based systems. The extensibility of the micro-broker allows the integration of features available on resource-rich computing environments.

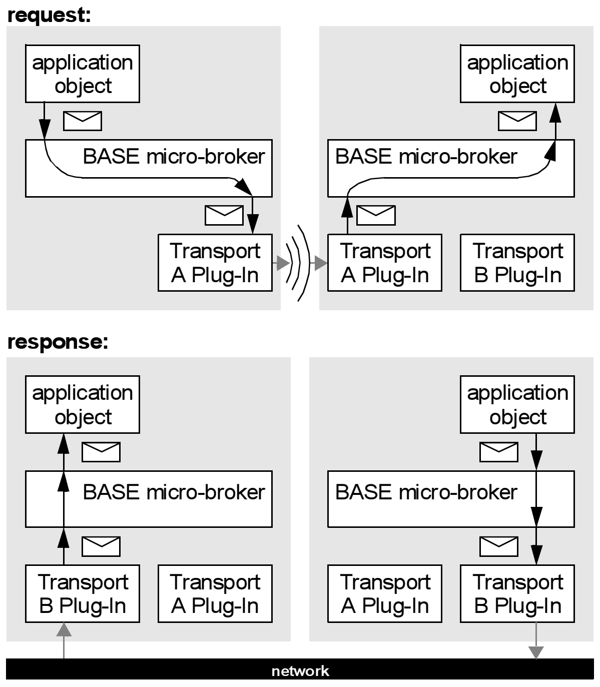

**Figure 1. Request / response interaction in BASE**

## 5. PCOM

The functionality provided by BASE offers a basic abstraction to ease application development. Still, additional mechanisms on top of BASE are needed to enable the automatic adaptation of applications during runtime in order to react to the changing availability of services or device capabilities according to the current application execution policy, e.g. to minimize the energy usage or to maximize the dependability of an application.

To achieve this, we propose an application model based on a component system (named PCOM). The application model specifies the architectural building blocks (modelled by components) and their interdependencies (modelled by contracts between components). At runtime, this specification is mapped to a concrete set of component instances where all mandatory contracts are fulfilled. Hence, PCOM-components offer a distinct functionality via contractually specified interfaces (following the definition of [16]). The functional properties of the contract are modelled in the interface itself whereas additional properties, e.g. dependency on another component, QoS requirements, or behavioral contracts via pre- and post-conditions, are explicitly modelled as contract types. This concept has been proposed in the realm of traditional component systems, e.g. [4, 17]. Contract types are templates for contract instances as well as the components are templates for component instances. When components are instantiated, contract types are mapped to concrete contracts which either offer the desired property, e.g. negotiate a distinct QoS property or bind to another component, or indicate a contract violation. An application is modelled via a special component (the socalled application anchor) which specifies the set of necessary sub-components. These components depend on each other according to the specified contract types.

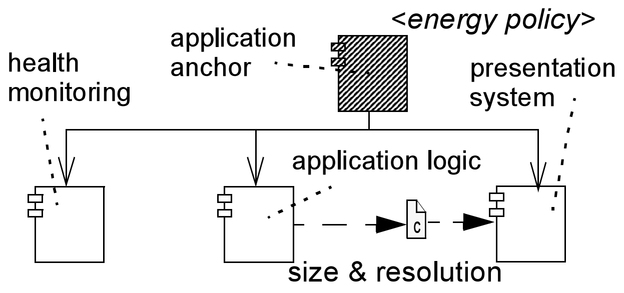

**Figure 2. A health monitoring application in PCOM.**

A simplified example for this is given in Figure 2. Here, a health monitoring application is shown, which outputs information and advices whenever a suitable display is in the vicinity. This application is formed by its application-anchor and three sub-component instances:

The *health monitoring* component is used to retrieve sensor information such as blood pressure, pulse, etc.

The *presentation system* component is responsible for presenting advices for certain health conditions, e.g. to calm down, take a distinct kind of medicine, on a display nearby.

The *application logic* component depends on these two components. It receives sensor information from the health monitoring component and derives advices, which it sends to the presentation system component. For simplicity, only the dependency between the application logic and the presentation component is shown. It is modelled as a contract which requires distinct size and resolution of the presentation system.

Additionally, a policy regarding energy consumption is shown, which is assigned to the application.

The application specification has to be mapped to instances on devices which realize the components. The different components are mapped to specific services residing on potentially different devices. Contracts between components have to be negotiated when a binding is established. For an example, before using or acquiring the display component a negotiation ensures that the resolution and size fulfill the contract.

The policy specifying the energy consumption is taken into account by the underlying framework when tasks that need a lot of energy, e.g. performing calculations or accessing remote components, are executed. The policies lead to configurations of the underlying BASE which will enforce them, e.g. in selecting the transport requiring the least energy.

The mapping of application policies, the contracts, and the binding of components deployed across different devices shall be provided by the framework. Currently, we have implemented BASE and designed the above sketched application model. Our next steps involve the design of the underlying framework as well as the mapping of contracts and policies to the services and mechanisms provided by BASE.

The overall framework will allow adaptation of applications by activating those applications where the application anchor contract is satisfied. That is, all dependencies of the applications can be fulfilled according to the application policies and contracts involved. Adaptation is supported by the mechanisms of the underlying middleware and the selection of alternative contracts. The execution context of an application is determined by the services available on nearby devices and the associated component instances from the application specification.

## 6. Related Work

### 6.1. Middleware Systems

In the past, a multitude of different middleware systems has been developed (e.g. [8, 15]) shielding application programmers not only from distribution of services but also different operating systems or hardware architectures. Conventional middleware systems are designed for mostly stable environments, in which service unavailability can be treated as an error, making these systems unsuitable for spontaneous networking environments.

The latter can be achieved by extending conventional middleware systems to dynamically reconfigurable middleware systems (e.g. [1, 5, 12]), which are able to adapt their behavior at runtime, e.g. how marshalling is done. Still, most existing reconfigurable middleware systems concentrate on powerful reconfiguration interfaces and not on supporting small, resource-poor devices.

The resource restrictions of such devices prohibit the application of a full-fledged middleware system. One way to address this is to restrict existing systems and provide only a functional subset (e.g. [9, 13]) leading to different programming models or a subset of available interoperability protocols. Another option is to structure the middleware in multiple components, such that unnecessary functionality can be excluded from the middleware dynamically. One example is the Universally Interoperable Core (UIC) [12]. Like BASE, UIC is based on a micro-kernel that can be dynamically extended to interact with different existing middleware solutions. However, different communication models or different protocols for outgoing and incoming messages are not supported.

### 6.2. Component Systems and Pervasive Computing

Component systems strive for independence of software components from underlying platform properties in order to allow their re-use. One way to achieve this is to model explicit context-dependencies, e.g. via contracts between components or contracts between the component container, such as in J2EE [14]. Typically, the inter-component contracts can be negotiated and various solutions exist, to ease the integration into the application framework, such as the aspect-oriented programming paradigm [1, 3]. While such approaches can be appropriately used to handle the inter-component contracts the component-container contract typically relies on a fixed common abstraction, making it unfeasible for pervasive computing environments where the container contract can change.

In the realm of ubiquitous computing the first approaches for component based systems are emerging. While Pebbles [10] is at a stage where it is hard to judge which requirements will be met, the Aura project [6] proposes a component framework similar to ours. The resource dependency of the Aura system is not addressed by the underlying middleware but by hand tailored resource monitors. Hence, only a comprehensive support of adaptation at the application layer, not on the middleware layer, is intended. Similar to Aura, One.world [7] and the Gaia system [11] shift the complexity of application adaptation to the programmer. Support of the underlying middleware is only provided with respect to communication issues.

## 7. Conclusion and Outlook

Pervasive computing environments differ from existing ones in the increasing heterogeneity of devices and networks. The spontaneous networking leads to situations, which are treated as errors in classical computing, but require distinct precautions since they can happen regularly. Based on typical scenarios we have derived a system model for pervasive computing and the support from system software and application adaptation. We have presented an extensible middleware platform which already provides basic abstractions to ease application development. The automatic adaptation of applications should be supported by a component model based on a framework. The basic abstractions of our middleware BASE can be used to build a framework for a component model. A contract concept is not only used to specify required properties for component interaction but also to indicate application configurations leading to a component-based application model. Adaptation of applications is reduced to validating required contracts and activating applications where all contracts are fulfilled. Contract enforcement and mechanisms to adapt are provided by BASE.

Currently, we have designed and implemented BASE. We are building prototypes for applications using BASE in order to gain experience on how the framework can support our application model. In the next steps of our work we will aim at completing the framework.

## References

[1] C. Becker and K. Geihs. Generic QoS-support for CORBA. In *Proceedings of 5th IEEE Symposium on Computers and Communications (ISCC'2000)*, 2000.

[2] C. Becker, G. Schiele, H. Gubbels, and K. Rothermel. Base - a micro-broker-based middleware for pervasive computing. In *Proceedings of the IEEE international conference on Pervasive Computing and Communications (PerCom)*, Mar. 2003.

[3] L. Bergmans and M. Aksit. Composing crosscutting concerns using composition filters. *Communications of the ACM*, 44(10), Oct. 2001.

[4] A. Beugnard, J.-M. Jzquel, N. Plouzeau, and D. Watkins. Making components contract aware. *IEEE Computer*, 13(7), July 1999.

[5] G. S. Blair, G. Coulson, P. Robin, and M. Papathomas. An architecture for next generation middleware. In *Proceedings of Middleware 2000*, 2000.

[6] S.-W. Cheng, D. Garlan, B. Schmerl, J. P. Sousa, B. Spitznagel, P. Steenkiste, and N. Hu. Software architecture-based adaptation for pervasive systems. In *International Conference on Architecture of Computing Systems (ARCS'02): Trends in Network and Pervasive Computing*, Apr. 2002.

[7] R. Grimm, T. Anderson, B. Bershad, and D. Wetherall. A system architecture for pervasive computing. In *Proceedings of the 9th ACM SIGOPS European Workshop*, Sept. 2000.

[8] Object Management Group. The common object request broker: Architecture and specification, revision 3.0, July 2002.

[9] Object Management Group. Minimum CORBA specification, revision 1.0, Aug. 2002.

[10] Oxygen System Group Homepage. http://o2s.lcs.mit.edu/.

[11] M. Román and R. H. Campbell. GAIA: Enabling active spaces. In *Proceedings 9th ACM SIGOPS European Workshop*, Sept. 2000.

[12] M. Román, F. Kon, and R. H. Campbell. Reflective middleware: From your desk to your hand. *IEEE Distributed Systems Online Journal, Special Issue on Reflective Middleware*, 2001.

[13] M. Román, A. Singhai, D. Carvalho, C. Hess, and R. Campbell. Integrating PDAs into distributed systems: 2K and PalmORB.

[14] Sun Microsystems. Java 2 platform, enterprise edition. http://java.sun.com/j2ee.

[15] Sun Microsystems. Java remote method invocation specification, revision 1.8.

[16] C. Szyperski. *Component Software - Beyond Object-Oriented Programming*. Addison-Wesley, 2nd edition, 1998.

[17] T. Weis, C. Becker, K. Geihs, and N. Plouzeau. An UML meta-model for contract aware components. In *Proceedings of UML 2001*, 2001.