3PC: System Support for Adaptive Peer-to-Peer Pervasive Computing

MARCUS HANDTE, Universität Duisburg-Essen
GREGOR SCHIELE, VERENA MATJUNTKE, and CHRISTIAN BECKER,
Universität Mannheim, Germany
PEDRO JOSÉ MARRÓN, Universität Duisburg-Essen

A major characteristic of pervasive computing applications is their ability to adapt themselves to changing execution environments and physical contexts. In this article, we analyze different kinds of adaptations and introduce a multidimensional classification for them. On this basis, we propose a novel approach for peer-to-peer-based pervasive computing that provides support for the identified classes and integrates them in a multilevel architecture. We give a comprehensive overview of this architecture and its current realization in the Peer-to-Peer Pervasive Computing (3PC) project, discussing what adaptation is realized on each level, how the levels interact with each other, and how the overall system benefits from the integrated treatment of adaptation.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems; D.4.7 [**Operating Systems**]: Organization and Design; D.2.7 [**Software Engineering**]: Distribution Maintenance and Enhancement

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Adaptation, automation, pervasive computing, peer-to-peer, multilevel adaptation, system software

ACM Reference Format:
Handte, M., Schiele, G., Matjuntke, V., Becker, C., and Marrón, P. J. 2012. 3PC: System support for adaptive peer-to-peer pervasive computing. ACM Trans. Autonom. Adapt. Syst. 7, 1, Article 10 (April 2012), 19 pages.
DOI = 10.1145/2168260.2168270 http://doi.acm.org/10.1145/2168260.2168270

1. INTRODUCTION

As envisioned by Mark Weiser [1991], the overall goal of pervasive computing is to provide seamless and distraction-free support for the everyday tasks of users through computer technology. To achieve this goal, pervasive computing foresees the integration of miniaturized computers into everyday objects. Using wireless communication technology the devices can form networks spontaneously. Through sensors, they are able to perceive their physical environment, that is, their *context*,

This work has been partially supported by CONET (Cooperating Objects Network of Excellence) and PECES (Pervasive Computing in Embedded Systems), both funded by the European Commission under FP7 with contract numbers FP7-2007-2-224053 and FP7-224342-ICT-2007-2 respectively.
Authors' addresses: M. Handte (corresponding author), Networked Embedded Systems Group, Department of Computer Science and Applied Cognitive Sciences, Universität Duisburg-Essen, Duisburg, Germany; email: marcus.handte@uni-due.de; G. Schiele, V. Matjuntke, and C. Becker, Information Systems II, Department of Business Administration, Universität Mannheim, Mannheim, Germany; P. J. Marrón, Networked Embedded Systems Group, Department of Computer Science and Applied Cognitive Sciences, Universität Duisburg-Essen, Duisburg, Germany.

DOI 10.1145/2168260.2168270 http://doi.acm.org/10.1145/2168260.2168270

and they can recognize user activities autonomously. This allows implicit interactions between devices and users which can improve task support.

Pervasive computing introduces a new class of networked computer systems. Due to integration, devices are highly *heterogeneous*, ranging from small, resource-poor embedded systems to resource-rich general-purpose computers. Due to the fact that many everyday objects are mobile and use of short-range communication, the overall system is highly *dynamic*. Finally, since the typical lifespan of an everyday object exceeds the lifespan of a computer, pervasive systems *evolve* continuously.

Pervasive applications must address these characteristics to be available at any time, any place. The main approach to do so is to *adapt* the application at runtime to different devices, contexts, and environments. Usually, it is not feasible to push the responsibility for adaptation to the user as this causes additional distraction. As a consequence, the development of pervasive applications is often challenging, since the application must adapt with little to no user interaction. System software for pervasive computing can mitigate this by providing generic adaptation support.

In this article, we present a novel multilevel system software that supports adaptation in a modular yet efficient manner. This system software is developed in the Peer-to-Peer Pervasive Computing (3PC) project since 2002 and comprises work undertaken in a number of subprojects since then. 3PC establishes dynamic device groups, so-called smart peer groups, at runtime and allows executing distributed adaptive applications on them. In contrast to other approaches, 3PC does not require any external infrastructure for this, for example, in the form of a smart environment. Instead, nearby devices directly discover and connect with each other.

3PC offers extensive support for different kinds of adaptation via a number of subsystems. Previously, we presented each of these subsystems in detail. In this article, we concentrate on the architectural aspects, that is, the subsystem structure itself. We give a brief overview of each subsystem and describe how they jointly realize comprehensive adaptation support. Thereby, we make the following contributions.

(1) We present a classification for adaptation support that can be used to identify possible adaptation subsystems and to specify their responsibilities.
(2) We describe how the different subsystems of 3PC realize the various adaptations at different levels to provide comprehensive support for automatic adaptation.
(3) We show that providing integrated system support for application adaptation at different levels can greatly improve the overall efficiency of the system.

The remainder of this article is structured as follows. In the next section, we introduce a classification for adaptation support in pervasive computing and outline associated design considerations. In Section 3, we discuss adaptation support in the 3PC infrastructure. In Section 4, we show that providing adaptation support at various levels of the software stack can greatly improve the performance of an adaptive system. In Section 5, we contrast the 3PC approach with related work on the basis of the classification introduced in Section 2. In Section 6, we conclude the article with a brief summary and a discussion of open challenges.

2. ADAPTATION IN PERVASIVE COMPUTING

The dynamics and heterogeneity of pervasive systems as well as the fact that pervasive applications are usually distributed raises an inherent need for adaptation. Clearly, adaptation can be reduced by minimizing the dependencies of an application on its execution environment. In many cases, however, such a reduction leads to a significant loss in quality. In fact, pervasive computing demands applications that leverage the available devices thoroughly to achieve its goal.

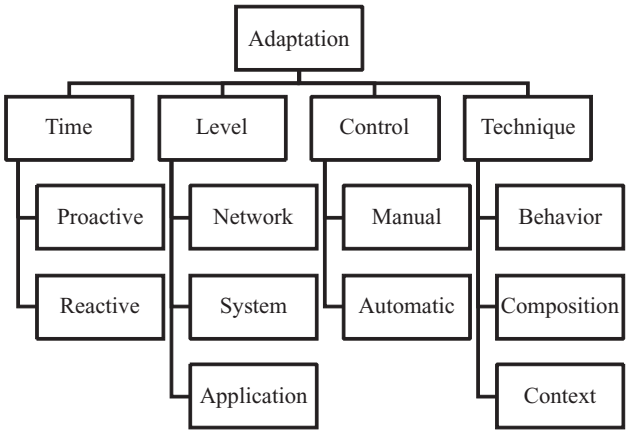

Fig. 1. Adaptation space in pervasive computing.

Pervasive applications can be modeled as a set of application and system services that are executed on multiple (potentially mobile) devices and interact with each other using a set of networking services. Thus, adaptation can occur in different ways. Moreover, it is possible to classify existing systems from different perspectives. For example, from an algorithmic perspective, we may classify adaptation according to synchronization requirements or convergence characteristics, etc.

In the following, we classify adaptation support for pervasive applications from an architectural perspective. The resulting dimensions are adaptation time, level, control, and technique as depicted in Figure 1. Next, we discuss each of them. A comprehensive classification of existing systems along the dimensions is presented in Section 5 and summarized in Figure 8.

2.1. Adaptation Time

The first dimension classifies adaptation regarding the point in time when an adaptation takes place. Here, we can differentiate proactive and reactive adaptation.

—*Proactive:* Proactive adaptation denotes modifications of an application performed before an application can no longer be executed. Thus, it tries to avoid application failures. To enable this kind of adaptation, the entity that controls the adaptation process needs a priori knowledge about future system states, for example, by using a prediction heuristic.
—*Reactive:* Reactive adaptation denotes a modification to an application that takes place at a point in time when the application can no longer be executed, for example, due to a lack of resources. As such, it fixes an application when it has already experienced a condition that prevents execution. For this, the entity controlling the adaptation needs to be able to detect failures whenever they occur.

From a user's perspective, proactive adaptation is usually preferable over reactive adaptation as it results in an uninterrupted execution. From a system developer's perspective, however, proactive adaptation is only possible with sufficiently accurate predictions for future states of the system. Due the difficulty of developing such predictions for complex systems that involve users, most existing systems including 3PC are focusing primarily on reactive adaptation.

2.2. Adaptation Level

The second dimension classifies adaptation with respect to the adaptation level, that is, the level of the software stack at which adaptation takes place. Based on the system model there are three possibilities that can occur either isolated or in combination.

—*Network:* The first possibility is to perform adaptation at the network level, that is, the level that connects multiple application and/or system parts with each other. Examples for this are a handover between different communication technologies or a dynamic modification of a compression factor during a transmission.
—*System:* The second possibility is adaptation at the system level which may adapt basic services. As an example, a context service may migrate data dynamically depending on the access pattern. Similarly, a discovery service might dynamically cluster devices to reduce lookup latency.
—*Application:* The third possibility is to perform adaptation at the application level by changing the composition of the application or by changing its behavior. As an example, the user interface of an application might be switched from one device to another.

Although it seems intuitive to handle changes solely at the lowest possible level, in practice, pursuing this approach can be inefficient or even ineffective. For example, instead of adapting to weak connectivity it can be beneficial to adapt an application structurally. The opposite approach, that is, pushing responsibility to the upper levels, does not result in viable support either, as it leads to coupled system structures that are hard to debug and maintain. 3PC applies an intermediate solution where lower levels provide generic mechanisms that can be fine-tuned by upper-level policies.

2.3. Adaptation Control

The third dimension classifies adaptation based on the control of the adaptation process. The control can be classified into manual and automatic.

—*Manual:* Manual adaptation is performed by a human. Beyond supporting manual modifications, the person that controls the adaptation must be supplied with a mental model of the application. Furthermore, the person must be able to perceive relevant properties to make proper adaptation decisions.
—*Automatic:* Automatic adaptation is performed by the application without user intervention. For this, the application must possess the same type of information that is required for manual adaptation. In addition, however, automatic adaptation requires a mechanism that selects and executes the modifications.

The goal of providing distraction-free support for tasks limits the applicability of manual adaptation, but pushing the responsibility for adaptation into the application complicates development. Selecting an action in response to a change requires reasoning about effects on the execution environment. However, due to the continuous evolution of pervasive systems, it is hard to foresee all possible states and effects at development time. Thus, this reasoning must be performed at runtime which complicates testing. As mitigation, the responsibility for adaptation can be pushed into the system software. Yet, enabling the system software to adapt an application requires explicit application knowledge which can often only be acquired by forcing applications to adhere to a particular application model.

2.4. Adaptation Technique

Finally, the fourth dimension classifies adaptation depending on the type of modification applied to the interacting parts that constitute the application. In this dimension, we can identify behavior, composition, and context adaptation.

—*Behavior:* Behavior adaptation modifies the way in which a certain part of the application is realized, for example, by changing parameters. Usually, behavior adaptation is applied to functionalities that are inherently configurable such as media transcoding, etc.
—*Composition:* Composition adaptation modifies an application by changing the structure or the distribution of the functionality that constitutes the application. This may entail simple isomorphic changes such as migration or more complex forms of recomposition.
—*Context:* Beyond adapting itself, a pervasive application can adapt its physical context. Obviously, this requires some way to actually change context, for example, using adequate actuators. In addition, changing the context must be coordinated with other applications to avoid oscillating changes.

Intuitively, the set of adaptation techniques that can be used to compensate for a particular change is constrained by the granularity of control that is available. For example, behavior and composition adaptation require parametrizable or modular applications, respectively. Furthermore, the set is also constrained on the basis of the type of change that shall be handled. For example, behavior adaptation is usually not applicable to service unavailability. On top of that, each of the techniques usually results in vastly different adaptation costs. While behavior adaptations are often fast, adapting the composition or context can take significantly longer. As a result, 3PC applies behavior, composition, and context adaptation at different levels. The goal thereby is to provide an effective and efficient combination.

3. ADAPTATION IN THE 3PC PROJECT

In the following, we discuss how adaptation is supported in the Peer-to-Peer Pervasive Computing (3PC) project. The 3PC project provides a software infrastructure to support adaptive pervasive computing applications. Its main goal is to fully automate adaptation, relieving application developers from implementing adaptation logic. To do so, 3PC follows a multilevel approach, supporting adaptation on all three levels of the software stack, the network, the system, and the application level, thereby applying behavior, composition, and context adaptation.

3.1. The 3PC Software Infrastructure

3PC follows a peer-to-peer system model which we refer to as the *smart peer group* model. The basic idea is that the devices present in a user's physical environment configure themselves automatically into a smart peer group which can execute distributed applications. All necessary resources and system services are provided jointly by the devices in the group. This enables the execution of pervasive applications everywhere. In contrast to this, other system software (e.g., Román and Campbell [2000], Garlan et al. [2002], and Johanson et al. [2002]) primarily focuses on a *smart environment* system model where one or more dedicated servers are providing basic system services within a spatially restricted area. In comparison to smart environments, smart peer groups are much more dynamic, as any device may leave the user's proximity at any time. Thus, we cannot assume that any device is always available to provide a service but we must prepare for its possible sudden loss.

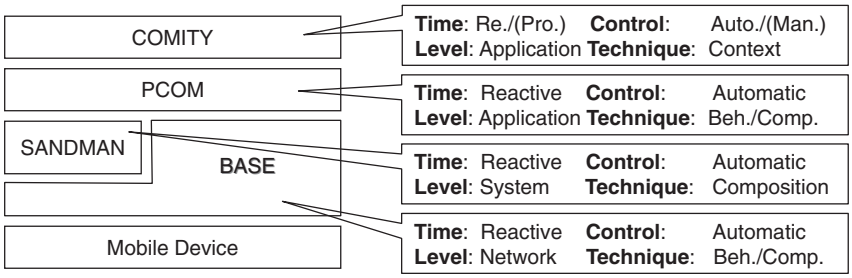

Fig. 2. 3PC software infrastructure overview.

The 3PC software infrastructure is composed of several subsystems as shown in Figure 2 which reduces the implementation complexity. The communication middleware *BASE* provides basic networking functionality, most notably remote service invocation. It supports adaptation on the network level to optimize the communication quality and reachability of remote services. *SANDMAN* extends BASE with adaptation on the system level to reduce the energy consumption of mobile devices. To do this, SANDMAN automatically clusters the devices within a smart peer group around a cluster head that is automatically (re-)elected. Thereby, it removes redundant system services and adapts the sleep schedules of idle devices to minimize possible usage delays. To handle persistent failures that cannot be masked, *PCOM* adds support for adaptation on the application level. PCOM introduces a minimal component system and an associated set of configuration algorithms and adaptation mechanisms that can automatically configure and adapt a distributed application at runtime. Finally, if it is necessary to adapt multiple applications consistently, *COMITY* provides a cross-application coordination framework that performs context adaptation. The framework identifies conflicting context influences of different applications and tries to resolve them using configurable resolution strategies. If no automatic resolution can be found, COMITY falls back to manual adaptation.

Together, the 3PC subsystems employ adaptation on all levels. Thereby, the focus lies on automatic adaptation which is transparent to both the application developer as well as the user. With the exception of COMITY which applies reactive and proactive adaptation, the remaining subsystems are solely reactive. This ensures broad applicability without requiring accurate predictions and, as discussed in Section 4, it can greatly improve the overall efficiency. Next, we describe the subsystems and their interaction in more detail.

3.2. Network-Level Adaptation: BASE

BASE [Becker et al. 2003] is an object-oriented communication middleware that has been tailored towards the specific requirements of pervasive systems. As such, BASE provides networking functionality for peer-to-peer interaction. To cope with the heterogeneity and evolution of communication technologies and protocols, BASE relies on a microbroker design that introduces a minimum core. This so-called invocation broker can be extended with plug-ins on a per-device basis. As shown in Figure 3, the core introduces proxy objects to provide a uniform programming interface to system services and application objects. Furthermore, it relies on a plug-in manager to perform remote communication.

At runtime, the microbroker is responsible for dividing service and object interactions into individual invocations, that is atomic units of transmission. These invocations are then mediated and synchronized according to the desired interaction pattern. To perform the mediation for local interactions, the invocation broker makes use of an object

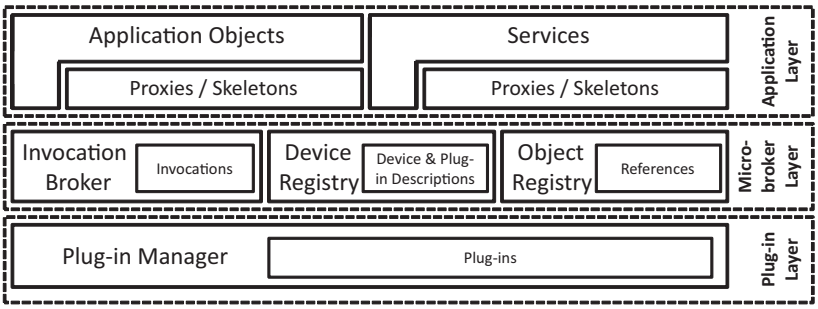

Fig. 3. BASE architecture.

registry. To perform remote mediation, the invocation broker forwards the invocations to the plug-in manager. The plug-in manager then uses the plug-ins to take care of all remaining transmission tasks such as (de-)marshalling, compression, encryption, transmission, reception, etc.

To support a broad range of applications with a small set of plug-ins, a single plug-in usually handles only one aspect of communication. This enables the plug-in manager to adapt the communication stack flexibly. As explained in Section 2.4, this resembles an instance of composition adaptation. Since different network technologies and protocols often posses tuning parameters such as compression rates or retransmission counters, a plug-in may expose them to support behavior adaptation. This reduces the number of plug-ins and thus, it speeds up composition.

As discussed, the basis of composition adaptation is structural knowledge. Since the tasks performed by plug-ins are known in advance, it is possible to uniformly classify plug-ins statically. This is done by means of a plug-in framework that defines several layers of plug-ins [Handte et al. 2010]. Using structural knowledge about the layers, the composition of a communication stack is performed automatically by the plug-in manager. Thereby, the plug-in manager ensures that the resulting stacks are functionally complete. In addition, it also ensures that the communicating devices are equipped with compatible sets of plug-ins. To do this, each plug-in is equipped with a plug-in description that models its functionality and compatibility. This description is then proactively distributed (and updated) among the devices as part of the device discovery process.

To provide control over the composition process, the plug-in manager exposes a configuration framework to the higher layers. This framework enables application developers to formulate requirements on individual protocol layers. Specifically, an application developer may request the presence of a certain protocol or the mandatory usage of a particular communication technology for transmission. In addition, the application developer may also specify parameters and possible relaxations.

To compose a stack, the plug-in manager relies on a backtracking algorithm that exhaustively searches through all possible combinations. If the composition succeeds, the stack configuration can be used to initiate the communication. If no satisfactory stack can be found, either the requested parameter set cannot be realized or the remote device is not available at this time. BASE does not handle such failures. Instead, the plug-in manager signals them to higher system layers using exceptions, enabling them to initiate other types of adaptation. As an example, PCOM may use this information by adapting the application composition such that the remote device is not used at all. In this way, BASE tries to shield higher layers from adaptations as much as possible by performing comparatively inexpensive network-level adaptations if possible.

3.3. System-Level Adaptation: SANDMAN

The SANDMAN [Schiele et al. 2004] subsystem performs adaptation on the system level by adapting system services. The goal of SANDMAN is to reduce the redundant provisioning of system services using automatic adaptation by composition. Due to the dynamic nature of smart peer groups, devices cannot rely on the availability of a particular device at any point in time. Thus, important system services must be provided locally by all devices to ensure continuous availability. Intuitively, this often leads to a high number of redundant services. To mitigate this, SANDMAN dynamically identifies a subset of devices that are sufficient to provide a service reliably. All other devices can stop providing the system service locally which enables them to enter a low-power mode to save energy.

In the following, we describe how SANDMAN can be used to adapt the Service Discovery System Service (SDSS) but SANDMAN can be extended to support other system services as well. A Service Discovery System Service (SDSS) provides up-to-date information about the functionalities that are currently available in an environment. To do so, it exchanges descriptions of client needs and service offers between the participating devices. A suitable SDSS must be able to discover services promptly and precisely and it can have a *mediator-based* (e.g., uddi.org [2004]) or a *peer-based* (e.g., Sun Microsystems [2001]) system organization.

A SANDMAN-based SDSS provides an adaptive alternative between the two. It offers two different modes of operation. If the system environment is highly dynamic, SANDMAN keeps devices activated to ensure that they can be discovered accurately and with little latency. In such environments, a mediator-based system organization would result in a large overhead. Therefore, the system uses a system organization in which all devices are equal peers and discovery requests are answered by each device directly. If SANDMAN identifies a more stable network topology, it switches to a mediator-based organization in which devices are elected to operate as LookUp Service (LUS). This is realized by organizing the devices into clusters.

A cluster consists of one Cluster Head (CH) and an arbitrary number of clustered devices (CN). Both CH and CNs can offer and use services. In order to save energy CNs periodically deactivate themselves when they are idle. After waking up, a CN waits for incoming client requests for the duration of a timeout interval. If no requests are received, it informs its CH and deactivates itself again. This way, unused CNs can save a substantial amount of energy. To provide timely discovery with low latencies despite the deactivated nodes, the CH stays active all the time. It acts as LUS managing all services running on devices in its cluster. Therefore, the CH can instantly answer any client's discovery request with a list of service descriptions and wake-up times. Note that each CN must check with the CH periodically. Otherwise, a CH could announce a service that is no longer available, for example, due to mobility. The CH's responsibility to stay awake is the trade-off taken in SANDMAN between energy-saving and discovery latencies.

SANDMAN can be divided into three parts. The first part, the cluster management, is responsible for creating and maintaining the clusters. It determines which nodes act as CH and assigns nodes to certain clusters. It also detects changes in node connectivity and adapts the system accordingly by reelecting CHs and reassigning nodes. To achieve stable clusters SANDMAN aims at clustering devices with similar mobility patterns, that is, devices moving together. This can be achieved by comparing the movement vectors of devices similar to Wang and Li [2002]. If device locations are unknown, we can use a simple heuristic to detect group mobility based on the duration that two devices have already stayed in each other's vicinity. The longer two devices can communicate, the more likely they have similar mobility patterns. Thus, we introduce a clustering delay threshold value that denotes the minimal duration of two nodes in each other's

communication range before they can be clustered. By changing the threshold, the cluster management protocol can be adapted to different intended stability levels at runtime. The second part, the service management, defines how services are registered at a CH and how clients perform lookups. The third part, the energy management, identifies idle CNs and schedules and executes their deactivation. To do so, it continuously monitors the state of each device, detects unused CNs, and schedules their deactivation periods. Scheduling is done on the cluster level, that is, for all CNs in a cluster, by the CH. The CH collects state information from all CNs, calculates a schedule for the whole cluster, and sends back deactivation commands and durations. The exact scheduling strategy can be selected by each CH autonomously to meet different goals. For optimal results, the three parts cooperate, for example, to schedule sleep times depending on client requests. A complete description can be found in Schiele [2007].

3.4. Application-Level Adaptation: PCOM

The PCOM subsystem [Becker et al. 2004] of the 3PC software infrastructure is targeting application-level adaptation. The need for application-level adaptation can be easily motivated by revisiting the scope adaptation performed by BASE and SANDMAN. With BASE and SANDMAN, a distributed application can be executed flawlessly as long as it is possible to mask communication failures by adapting a communication stack or changing a wake-up schedule. However, if connectivity cannot be restored by network- or system-level adaptation, the resulting failures cannot be masked. Instead, it is necessary to adapt the application, for example, by replacing a service or resource that has become unavailable with another one.

The PCOM component model differentiates between components and component instances. Components can be thought of as blueprints for their instances. The number of instances is a priori not restricted. Each component resides on exactly one device and each device runs a component container that creates and manages all local component instances. An instance provides a contractual description of its offered functionality and its requirements. The requirements can be split into local resource requirements and requirements on other components. The container assigns the local resources and, in cooperation with other containers, it instantiates and manages the required instances.

An PCOM application is a tree of component instances that is constructed by recursively starting the required instances of a root instance, the so-called application anchor. An application can only be started if all recursively required instances can be executed. If a required resource or component becomes unavailable at runtime, the application must be suspended until an executable configuration can be found or the application is stopped upon user request. It is noteworthy that a PCOM component container supports strictly limited resources, for example, to model exclusive resources. Strictly limited resources usually lead to a limitation on the number of instances that can be executed on a container. Since instances of different components can have overlapping resource requirements, creating an instance of one component can prohibit the instantiation of another one.

To enable external management of components and thus applications by means of the PCOM component container, components are bound to a life cycle that consists of a started, stopped, and paused state. Changes to the life cycle are triggered by the component container via callback methods that must be implemented by the component developer. To interact with required components, the component container provides each instance with local references to (possibly remote) component instances as well as local resources. By providing and managing the references, the component container ensures that they are resolved at runtime as long as a component instance is started. Once a required component instance becomes unavailable, the component container triggers the paused state recursively and initiates an adaptation. Once the adaptation

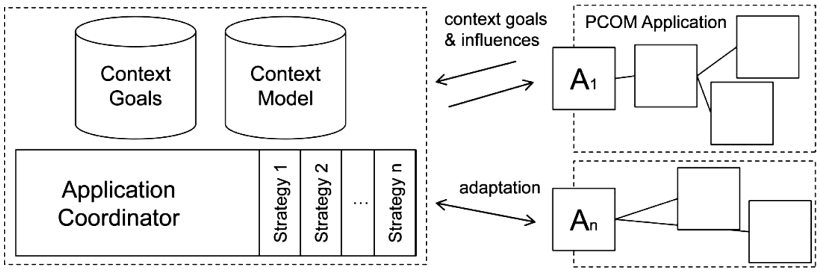

Fig. 4. Coordinated context adaptation with COMITY.

has been executed, the component instance is either stopped or started again. In the latter case, the component container has updated the references so that they are valid again.

The PCOM container is able to automatically determine executable configurations upon startup using a distributed backtracking algorithm [Handte et al. 2005]. Furthermore, on top of the signaling provided by BASE, it is also able to detect cases in which the application can no longer be executed and thus, needs to be adapted [Handte et al. 2007]. As introduced in Section 2.1, this type of adaptation is reactive since it performs adaptation after a failure occurred. In general, cases that require adaptation can be attributed to two types of changes, namely changes to preferences and changes to the available resources and devices. In PCOM, user preferences are expressed by means of contracts that describe the requirements on the application anchor. If a user changes these requirements, the component container can validate whether the running application satisfies the changed requirements by matching the modified contractual requirements with the current provision. Similarly, if the available set of resources changes dynamically, the change is usually reflected by means of a changed provision. Thus, if the modified provision no longer satisfies the current requirements, the application needs to be adapted.

If PCOM's configuration and adaptation algorithm is not able to compute an executable configuration, PCOM simply pauses the execution of the application until the required resources become available. Yet, in cases where multiple applications are executed, it might be possible to free up resources from other applications as an alternative. However, since such adaptations would require an coordinated treatment of multiple applications, they are not handled by PCOM directly but they are signaled to COMITY where they can be handled in an integrated manner.

3.5. Application-Level Adaptation: COMITY

The COMITY subsystem [Tuttlies et al. 2007; Majuntke et al. 2010] provides support for coordinated context adaptation. This form of adaptation is typically achieved via actuators available in the environment. For example, an ebook reader application whose current display requires a certain light level may switch on the light instead of selecting another display. In fact, context adaptation is available in diverse systems which integrate the use of actuators. However, context adaptation of multiple applications is usually not coordinated such that each application changes the shared context independently. This leads to interferences if applications have contradictory requirements which may cause cyclic changes.

To avoid cycles, COMITY provides a framework which coordinates the adaptation techniques of multiple applications in order to manage the occurring interferences. The relationship between COMITY and PCOM is shown in Figure 4. To enable automatic coordination, the framework requires each application to explicitly specify: (a) its

context influences, that is, how it modifies the context and (b) its context goals, that is, its requirements towards the shared context. For a PCOM application the context influences and context goals depend on the components which are used in the respective application configuration.

The context goals as well as the context influences are collected by a representative member of the smart peer group, the application coordinator. The selection of the member can be determined dynamically using the SANDMAN subsystem. Based on the collected information, the application coordinator determines a common context goal which considers the context goals of all concurrently executed applications. This common goal may change over time as applications are started or terminated in the smart peer group. If context goals of different applications contradict, COMITY may initiate a composition adaptation of a selected set of applications to adapt context goals which cause the contradiction.

For interference detection, the application coordinator continuously compares the constructed context goal with the current context. To do this, it must have access to a context model which is maintained by the members of the smart peer group. In addition to context provided by sensors, the context model holds all context influences which have been communicated by active applications. When the coordinator detects that the context goal is not met, either the context or active applications have to be adapted. The specific adaptation and the selection of applications which have to adapt depend on the resolution strategy which has been set for the application coordinator. Our current implementation of COMITY instructs those applications to adapt whose context influences violate the common context goal. To do this, COMITY marks the problematic PCOM components as unusable. This forces the component container to adapt the application. If no suitable solution can be found, COMITY can fall back to rudimentary manual adaptation support by asking the application users for a possible resolution. COMITY also provides basic support for proactive adaptation. To do so, it checks the context influences of PCOM components before instantiating them, instead of only checking the resulting context after a new component is started. Thus, context interferences can be detected and resolved before the actual context influence occurs. However, we are currently working on further support for proactive adaptation.

3.6. 3PC Implementation, Tools, and Applications

To evaluate the abstractions and algorithms introduced by the individual subsystems of the 3PC infrastructure, we have implemented and refined them over the course of several years. During that time, we relied on a bottom-up approach to support continuous integration and testing. The basis was the first prototype of BASE in 2002. Due to the increasing pervasiveness of the Java language at that time, we based the implementation on J2ME. To support resource-poor devices such as mobile phones, the core abstractions of all subsystems comply to the J2ME CLDC specification which introduces considerable restrictions on the features provided by the virtual machine. At the same time, relying only on minimal functionality also greatly simplified the porting of 3PC to different devices.

In addition to the infrastructure itself, we developed a considerable set of tools. The tools can be classified into compile-time and runtime tools. The compile-time tools are mostly generative in nature and include for example stub generators for BASE proxies and PCOM components. Since we used IBM Websphere Device Developer which is an Eclipse-based suite of development tools during the early stages of the implementation, the compile-time tools are implemented as Eclipse plug-ins. The runtime tools include visualizations for PCOM's configuration and adaptation algorithms as well as user interfaces and system inspectors.

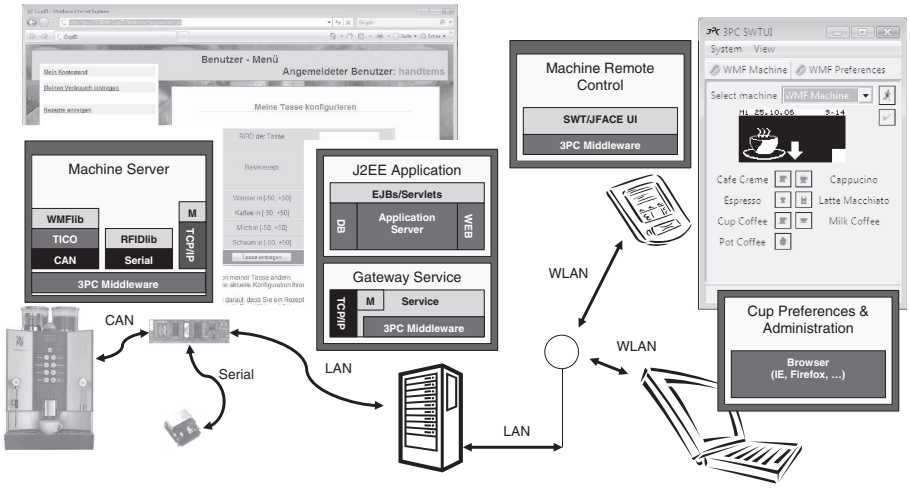

Fig. 5. Coffee maker application.

To validate the 3PC infrastructure, we implemented a number of application prototypes using the abstractions provided by BASE and PCOM. An example for PCOM is the Pervasive Presenter [Handte et al. 2006] which enables a user to display and control a PowerPoint presentation using a mobile phone. Due to the use of PCOM, the presenter is completely configuration free and neither the developer nor the user have to take manual steps to adapt the application. An example for a BASE application is the RFID-based billing system depicted in Figure 5. The coffee maker is equipped with a CAN bus to control the machine. A custom embedded system consisting of a TINI microcontroller and a Texas Instruments RFID reader connects to the CAN bus. The microcontroller runs an instance of BASE and exports a service via a LAN connection. When the RFID of a user is detected at the reader, the BASE service contacts an application server to retrieve the coffee preferences. Thereafter, the microcontroller issues the command to brew the desired coffee. Using an application running on a PDA, a service technician can retrieve information about the internal state of the machine.

At the time of writing, the developments on BASE and PCOM have been completed and they are freely available as open source under a BSD-style license (refer to http://www.3pc.info). We are currently using both systems as an application development platform in different research projects. Our current development effort focuses on the subsystem at the highest level, namely COMITY and the associated context management and application coordination services.

4. EVALUATION: OVERHEADS AND BENEFITS OF MULTILEVEL ADAPTATION

A key goal of the 3PC software infrastructure is to leverage adaptation on all levels of the software stack in order to provide thorough support for adaptive applications. In the following, we discuss and contrast the adaptation overheads induced by different levels. Thereafter, we discuss how the levels can be used for optimization.

4.1. Comparison of Overheads by Adaptation Level

At network level, BASE introduces composition and behavior adaptation in order to enable flexible communication support. Intuitively, computing a compatible protocol stack introduces additional delay experienced by the caller. This delay usually ranges in the order of several milliseconds. This can be seen on the left side of Figure 6, which depicts the total delay of a remote method call with different payload sizes

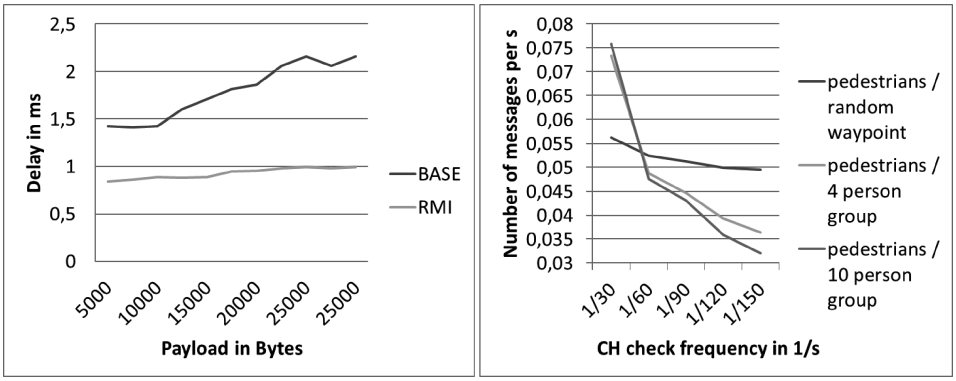

Fig. 6. BASE communication delay (left) and SANDMAN message overhead (right).

in BASE and RMI using two desktop PCs (Intel Core 2 Q6600, 2.4 GHz, 4GB RAM) connected via a switched Ethernet (1GBit). For both BASE and RMI, the depicted delay results from a null-call that has no server implementation. Obviously, the absolute delay depends on the computational capabilities of the device and thus, they may change with different hardware. However, in typical settings with WLAN or Bluetooth connectivity, the total communication delay is dominated by the transmission delay caused by the network. This even holds true for resource-poor devices such as PDAs whose Java implementations are less optimized. Furthermore, the latency could be reduced by caching stack compositions.

The adaptation overhead introduced by SANDMAN can be estimated in terms of message overhead. The right side of Figure 6 shows the resulting message overhead per second for different group sizes of pedestrians. For a useful evaluation, it is not sufficient to model mobility only with speed. Instead, we additionally must vary the number of nodes per group. We choose three characteristic group sizes: a single person moving randomly through town, a family moving together, and a tourist group. Each person carries one device with IEEE 802.11 wireless communication. A group size of one leads to the well-known random waypoint model. Families are modeled as groups of 4 devices, tourist groups contain 10 devices. In total, our experiments contain 40 devices moving in an area of 300m by 300m with up to 2 m/s. The overhead per device decreases for larger groups as cluster sizes increase. The main message overhead is caused by cluster creation and maintenance. Most notably, each CN periodically checks if its CH is still available using a request/response interaction with the CH. This is the CH check frequency. As can be seen, the CH check frequency largely influences the total message overhead. On the other hand, the higher the check frequency, the faster CNs detect a lost CH, which increases the service discovery quality. Another contributor to the continuous message overhead are regular CH announcements using hello messages as described earlier. For small groups this factor can dominate the overhead.

The application adaptation performed by PCOM is directly visible to the user. This is a result of the fact that PCOM performs adaptation reactively. Thus, if a failure is detected and fixed by the component container, the user will experience an interruption in the execution of the application. The configuration algorithms utilized in PCOM try to minimize the delay. However, configuration and adaptation can cause delays in the order of seconds depending on the number and type of the resource conflicts. Figure 7 shows the time taken to start and adapt an application with varying numbers of components organized in a binary tree. The tree has been distributed among the devices in such a way that no adjacent components reside on the same device and the

Number of Components	4	5	6	7	8	9	10	11	12
Start up	865	1152	1416	1472	1673	1923	2260	2613	2845
Adaptation	576	835	980	978	1178	1291	1593	1846	2038
PCOM + SANDMAN	617	769	826	852	1090	1154	1326	1512	1649

Fig. 7. PCOM adaptation delay (in ms).

devices have sufficient resources to execute all components. The measurements have been performed on 4 MDA Pro PDAs that are connected via IEEE802.11b. The devices utilize a greedy distributed configuration algorithm. In order to measure the adaptation performance, a contractual change is injected that can be resolved by changing other contracts. As a result, adaptation can be performed faster than start up. The resulting difference in latency is an indicator for the usefulness of support for both behavior and composition adaptation.

The overhead for application-level adaptation using COMITY cannot be measured systematically in terms of system parameters since it may entail user actions. For example, in order to change the context a user may have to move to a different location. Obviously, this overhead often weighs more than a delay caused by an adaptation mechanism. However, in some cases user involvement is the only solution and thus, it cannot be avoided. Yet, minimizing the amount of such cases is one of the optimizations enabled by 3PC's multilevel adaptation.

4.2. Benefits of Multilevel Adaptation

Probably the most obvious optimization results from the fact that the adaptation mechanisms at application level are using the adaptation mechanisms at network level. When comparing the overhead induced by BASE and PCOM, it is clearly visible that adaptations in BASE are several orders of magnitude faster. While protocol adaptation introduces delays of a few milliseconds, application adaptation can take seconds. Yet due to the fact that the PCOM component container and components are using BASE as communication mechanism, application adaptation can often be avoided since BASE can adapt communication if necessary.

A similar argument can be made by looking at the interaction between PCOM and COMITY. Context adaptation can be significantly more annoying to a user since it may require user actions. In contrast, adaptations performed by PCOM only cause interruptions. Thus, by relying on the adaptations performed by PCOM, the use of COMITY can often be avoided. In addition, the composition adaptation performed by PCOM also benefits from the context adaptation done in COMITY. In cases where resources are scarce, the computation of an application configuration can cause unacceptable delays. In such cases, COMITY can perform automated decisions that free resources, for example, by pausing or stopping low-priority applications.

Last but not least, there are also benefits from performing system-level adaptation with SANDMAN and automatic adaptation with PCOM in an integrated manner. The reason for this is that the configuration algorithms in PCOM are usually fully distributed to support the peer-to-peer system model. SANDMAN can reliably detect cases in which the networks are static and it automatically selects cluster heads to reduce the degree of distribution. This reduction of the degree of distribution can significantly improve the configuration delay of PCOM's configuration algorithms. The last row in Figure 7 shows the potential savings. When performing a simple contractual adaptation, computing a new configuration can already be improved by up to 20%. For 12 components this reduces the delay experienced by the user by almost half a second. In cases that require the resolution of conflicts, the advantage can be even higher.

5. RELATED WORK

Adaptation in pervasive computing has been researched by many groups. Figure 8 summarizes related approaches with respect to the classification given in Section 2. Most approaches only support a subset of the possible adaptation space. Others rely on the availability of a smart environment infrastructure. 3PC supports the full adaptation space, without the need of any predeployed infrastructure.

The most commonly supported adaptation technique is composition adaptation as provided in systems like MundoCore [Aitenbichler et al. 2007], PECES [Haroon et al. 2009], and IROS [Johanson et al. 2002], for example. Behavior adaptation is offered by some approaches, for example, Puppeteer [De Lara et al. 2001] and REFLECT [Schroeder et al. 2008]. Context adaptation is so far only supported by few systems, such as, REFLECT [Schroeder et al. 2008] and Vainio et al. [2008].

With respect to the different adaptation levels, adaptation on the network level is usually offered by communication middleware systems like Puppeteer [De Lara et al. 2001] and MundoCore [Aitenbichler et al. 2007]. Puppeteer supports behavior adaptation and automatically transcodes transmitted data depending on the available network resources. Reflective middleware systems, for example, UIC [Román et al. 2001], additionally allow application developers to adapt the behavior of system services at runtime. Further approaches like PARM [Mohapatra and Venkatasubramanian 2003] and PECES [Haroon et al. 2009] exclusively focus on system level by optimizing the execution of the system software itself. PARM relocates system services at runtime to remote devices. PECES assigns roles to dynamically form smart environments. Applications can build on top of these environments but must provide their own adaptation logic. Only the Runes middleware [Costa et al. 2005] supports adaptation on system and application level. It offers a minimal middleware core which can be extended by system and application components.

Several systems concentrate on adaptation on application level. Except for two approaches, all of them offer automatic adaptation support [Grimm et al. 2001; Herrmann et al. 2008; Mazzola Paluska et al. 2008; Ferscha et al. 2004] using composition adaptation. As an example, iROS [Johanson et al. 2002] assembles loosely coupled applications from components at runtime using an asynchronous EventHeap. REFLECT [Schroeder et al. 2008] builds applications from software components. By specifying provide and demand ports for each component, the middleware can construct applications dynamically. Additionally REFLECT offers behavior adaptation by chaining application components into control loops that automatically adapt their behavior. The two exceptions are Speakeasy [Want et al. 2003] and the approach of Vainio et al. [2008]. In contrast to all other approaches Speakeasy focuses on simplifying manual adaptation. As a second exception, Vainino et al. [2008] focus on a coordinated adaptation of the context and omit composition adaptation. Their approach learns the routines of a user in a smart home environment and uses this for automatic control.

The systems discussed so far focus on isolated adaptation levels or techniques. Yet, there are a number of projects that try to offer more comprehensive adaptation support on multiple levels and techniques. However, in contrast to 3PC, they realize smart environments and require preinstalled computer infrastructure.

Gaia [Román and Campbell 2000] provides adaptation on all levels using several subsystems. Acting as the underlying communication middleware, dynamicTAO [Roman et al. 1999] supports automatic network- and system-level adaptation. On top of this, Gaia structures adaptive applications using an extended Model View Controller model (MPCC). This model allows composition adaptation as defined by a system administrator. Automatic composition is added by the Olympus framework [Ranganathan et al. 2005] by mapping application specifications to resources.

Project		Time		Level			Control		Technique		
		pro	rea	app	sys	net	man	aut	beh	com	ctx
PUPPETEER			+			+		+	+		
MundoCore			+			+		+	+	+	
UIC			+		+	+		+		+	
PARM			+		+			+		+	
PECES			+		+					+	
RUNES			+	+	+			+		+	
ALLOW		+		+				+		+	
O2S		+		+				+		+	
P2PCOMP			+	+				+		+	
ONE.WORLD			+	+				+	+	+	
SPEAKEASY		na	na	+			+			+	
IROS		+		+				+		+	
REFLECT			+	+				+	+	+	(+)
Vainio et al.		+		+				+			+
GAIA	dynamic TAO		+		+	+		+		+	
	MPCC	na	na	+			+			+	
	Olympus		+	+				+		+	
AURA	Odyssey		+			+		+	+		
	Coda		+		+			+	+		
	SPEC-TRA		+	+				+		+	
	PRISM	+		+			+			+	
3PC	BASE		+			+		+	+	+	
	SAND-MAN		+		+			+		+	
	PCOM		+	+				+	+	+	
	COMITY	(+)	+	+			(+)	+			+

Fig. 8. System support for pervasive adaptation.

The Aura project [Garlan et al. 2002] supports behavior and composition adaptation on all three levels. On network level, Odyssey [Satyanarayanan 1996] adapts the quality of transferred data leading to a behavior adaptation. Coda [Satyanarayanan 2002] is located on system level. It realizes an adaptive distributed file system with support for disconnected operation. On top, Spectra [Flinn et al. 2001] executes application parts remotely depending on the current system environment. Finally, Prism [Sousa and Garlan 2002] manages user tasks by mapping task descriptions to specific applications in a smart environment. The currently required user tasks are derived automatically by estimating the user's intentions. This allows to proactively prepare adaptations before they become necessary. Concrete mappings are provided manually by system administrators.

6. CONCLUSION

The development of adaptive pervasive applications is a complex and error-prone process. To mitigate this, application developers need efficient system support for adaptation. In pervasive computing, adaptation should be supported automatically on different levels and using different techniques. In this article, we structured adaptation in pervasive computing from an architectural perspective and outlined design considerations. We presented the 3PC software infrastructure and discussed how it applies a broad set of adaptation techniques. 3PC offers adaptation support on all system levels, from network to system and application. Providing such an integrated solution eases development and it improves the adaptation performance.

The main future challenge for our work is to integrate support for proactive adaptation. Until now most systems, including our own, focus primarily on reactive adaptation. COMITY provides restricted proactivity by prohibiting applications to perform problematic context adaptations before they occur. However, this is only a first step towards real proactive adaptivity. First, we need to develop accurate prediction algorithms to foresee future scenarios for adaptation. Second, proactivity support is required on all levels to gain really seamless application execution. Otherwise, truly distraction-free pervasive computing will not be accomplishable.

REFERENCES

Aitenbichler, E., Kangasharju, J., and Mühlhäuser, M. 2007. Mundocore: A light-weight infrastructure for pervasive computing. *Pervas. Mobile Comput. 3,* 4, 332–361.

Becker, C., Handte, M., and Schiele, G. 2004. PCOM – A component system for pervasive computing. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communication.*

Becker, C., Schiele, G., Gubbels, H., and Rothermel, K. 2003. Base – A micro-broker-based middleware for pervasive computing. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communication.*

Costa, P., Coulson, G., Mascolo, C., Picco, G. P., and Zachariadis, S. 2005. The runes middleware: A reconfigurable component-based approach to networked embedded systems. In *Proceedings of the 16th International Symposium on Personal Indoor and Mobile Radio Communications*. 11–14.

De Lara, E., Wallach, D. S., and Zwaenepoel, W. 2001. Puppeteer: Component-Based adaptation for mobile computing. In *Proceedings of the (USITS'01) 3rd Conference on USENIX Symposium on Internet Technologies and Systems*. 14–25.

Ferscha, A., Hechinger, M., Mayrhofer, R., and Oberhauser, R. 2004. A light-weight component model for peer-to-peer applications. In *Proceedings of the 24th International Conference on Distributed Computing Systems Workshops*. 520–527.

Flinn, J., Narayanan, D., and Satyanarayanan, M. 2001. Self-Tuned remote execution for pervasive computing. In *(HOTOS '01) Proceedings of the 8th Workshop on Hot Topics in Operating Systems*. IEEE Computer Society, Washington, DC, 61.

Garlan, D., Siewiorek, D. P., Smailagic, A., and Steenkiste, P. 2002. Project aura: Toward distraction-free pervasive computing. *IEEE Pervas. Comput. Mag. 1,* 2.

Grimm, R., Davis, J., Lemar, E., MacBeth, A., Swanson, S., Anderson, T., Bershad, B., Borriello, G., Gribble, S., and Wetherall, D. 2001. Programming for pervasive computing environments. Tech. rep. UW-CSE-01-06-01, University of Washington. June.

Handte, M., Becker, C., and Rothermel, K. 2005. Peer-Based automatic configuration of pervasive applications. In *Proceedings of the International Conference on Pervasive Services*. 249–260.

Handte, M., Herrmann, K., Schiele, G., Becker, C., and Rothermel, K. 2007. Automatic reactive adaptation of pervasive applications. In *Proceedings of the International Conference on Pervasive Services*. 214–222.

Handte, M., Urbanski, S., Becker, C., Reinhard, P., Engel, M., and Smith, M. 2006. 3pc/marnet pervasive presenter. In *Proceedings of the 4th IEEE International Conference on Pervasive Computing and Communications (PerCom'06) Demos*.

Handte, M., Wagner, S., Schiele, G., Becker, C., and Marrón, P. J. 2010. The base plug-in architecture - Composable communication support for pervasive systems. In *Proceedings of the 7th ACM International Conference on Pervasive Services*.

Haroon, M., Handte, M., and Marron, P. J. 2009. Generic role assignment: A uniform middleware abstraction for configuration of pervasive systems. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communications*. 1–6.

Herrmann, K., Rothermel, K., Kortuem, G., and Dulay, N. 2008. Adaptable pervasive flows - An emerging technology for pervasive adaptation. In *Proceedings of the 2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops*. 108–113.

Johanson, B., Fox, A., and Winograd, T. 2002. The interactive workspaces project: Experiences with ubiquitous computing rooms. *IEEE Pervas. Comput. Mag. 1,* 2, 71–78.

Majuntke, V., Schiele, G., Spohrer, K., Handte, M., and Becker, C. 2010. A coordination framework for pervasive applications in multi-user environments. In *Proceedings of the 6th International Conference on Intelligent Environments (IE'10)*.

Mazzola Paluska, J., Pham, H., Saif, U., Chau, G., Terman, C., and Ward, S. 2008. Structured decomposition of adaptive applications. *Pervas. Mobile Comput. 4,* 6, 791–806.

Mohapatra, S. and Venkatasubramanian, N. 2003. PARM: Power aware reconfigurable middleware. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*. 312–321.

Ranganathan, A., Chetan, S., Al-Muhtadi, J., Campbell, R. H., and Mickunas, M. D. 2005. Olympus: A high-level programming model for pervasive computing environments. In *Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications*. 7–16.

Román, M. and Campbell, R. H. 2000. GAIA: Enabling active spaces. In *Proceedings of the 9th ACM SIGOPS European Workshop*. ACM, Press, New York.

Roman, M., Kon, F., and Campbell, R. H. 1999. Design and implementation of runtime reflection in communication middleware: The dynamictao case. In *Proceedings of the International Conference on Distributed Computing Systems*. 0122.

Román, M., Kon, F., and Campbell, R. H. 2001. Reflective middleware: From your desk to your hand. *IEEE Distrib. Syst. Online J. Special Issue on Reflective Middleware*.

Satyanarayanan, M. 1996. Mobile information access. *IEEE Personal Comm. 3,* 1, 26–33.

Satyanarayanan, M. 2002. The evolution of coda. *ACM Trans. Comput. Syst. 20,* 2, 85–124.

Schiele, G. 2007. System support for spontaneous pervasive computing environments. Ph.D. thesis, Universität Stuttgart.

Schiele, G., Becker, C., and Rothermel, K. 2004. Energy-Efficient cluster-based service discovery. In *Proceedings of the 11th ACM SIGOPS European Workshop*.

Schroeder, A., Zwaag, M. v. d., and Hammer, M. 2008. A middleware architecture for human-centred pervasive adaptive applications. In *Proceedings of the 2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops*. 138–143.

Sousa, J. P. and Garlan, D. 2002. Aura: An architectural framework for user mobility in ubiquitous computing environments. In *Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture*. Kluwer Academic Publishers, 29–43.

Sun Microsystems. 2001. Jini technology core platform specification, version 1.2. http://www.csag.ucsd.edu/teaching/cse291s03/Readings/core1_2.pdf.

Tuttlies, V., Schiele, G., and Becker, C. 2007. Comity - Conflict avoidance in pervasive computing environments. In *OTM Workshops (2)*. 763–772.

uddi.org. 2004. UDDI spec technical committee draft, version 3.0.2. online. http://uddi.org/.

Vainio, A.-M., Valtonen, M., and Vanhala, J. 2008. Proactive fuzzy control and adaptation methods for smart homes. *IEEE Intell. Syst. 23,* 2, 42–49.

WANG, K. H. AND LI, B. 2002. Group mobility and partition prediction in wireless ad-hoc networks. In *Proceedings of the IEEE International Conference on Communications (ICC)*.

WANT, R., PERING, T., AND TENNENHOUSE, D. 2003. Comparing autonomic and proactive computing. *IBM Syst. J. 42,* 1, 129–135.

WEISER, M. 1991. The computer for the twenty-first century. *Sci. Amer. 265,* 3, 94–104.

Received January 2010; revised June 2011; accepted July 2011