# A Framework for Adapting the Distribution of Automatic Application Configuration

Stephan Schuhmann
Universität Stuttgart
Institute of Parallel and Distributed Systems (IPVS)
Universitätsstr. 38
70569 Stuttgart, Germany
schuhmann@ipvs.uni-stuttgart.de

Klaus Herrmann
Universität Stuttgart
Institute of Parallel and Distributed Systems (IPVS)
Universitätsstr. 38
70569 Stuttgart, Germany
herrmann@ipvs.uni-stuttgart.de

Kurt Rothermel
Universität Stuttgart
Institute of Parallel and Distributed Systems (IPVS)
Universitätsstr. 38
70569 Stuttgart, Germany
rothermel@ipvs.uni-stuttgart.de

## ABSTRACT

Numerous current research projects deal with the issue of automatic application configuration in pervasive computing scenarios. While completely distributed configuration is inevitable in infrastructure-less ad hoc scenarios, many realistic pervasive application scenarios are located in heterogeneous environments where additional computation power of resource-rich devices can be utilized. However, most of the current projects either focus solely on smart environments which rely on additional infrastructure devices, or they address ad hoc environments and treat all involved devices as equal. This leads to suboptimal results in case of present resource-rich devices, as their additional computation power is not exploited. In this paper, we present a framework that is based on clustering and allows the efficient and flexible support of automatic application configuration both in infrastructure-based and ad hoc environments. This is realized by a new concept called *Virtual Container* that enables the local emulation of remote devices' functionalities and resources. In our evaluation results, we prove that this concept considerably reduces configuration delays noticeable by the user.

## Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed Applications*; D.2.11 [**Software Engineering**]: Software Architectures—*Domain-specific architectures*

## General Terms

Design, Measurement, Performance

## Keywords

Adaptivity, Components, Middleware, Pervasive Computing



## 1. INTRODUCTION

The research area of *pervasive computing* has emerged from the fact that many heterogeneous mobile end-user devices are available that have to be connected seamlessly with each other. This integration is possible by the introduction of *middleware* platforms that insert a logical layer between hardware and software. The middleware provides certain basic services to ease the development of distributed applications. Therefore, such platforms offer a unique programming model and try to hide the heterogeneity of underlying layers. Main goal of the middleware is technology abstraction and the increase of the quality of service for the users.

In this connection, applications for pervasive computing need to be configured before execution, as the availability of the needed resources may change over time. This configuration should be performed automatically without user intervention to enable technology abstraction. In order to additionally increase the quality of service for the user, automatic configuration should be calculated as fast as possible.

Regarding pervasive computing scenarios that support automatic application adaptation on system-level, there exist two groups: *Smart Environments*, and *Peer-to-Peer Pervasive Computing Environments*. *Smart Environments* are systems that provide a specific functionality by additional infrastructure devices. In such scenarios, there is usually an ongoing contact between the mobile devices and the environment. Contrary to this, *Peer-to-Peer Pervasive Computing* scenarios are based on spontaneously networked environments without infrastructure support. These ad hoc networks are typically highly dynamic, and the involved systems are acting in a cooperative way. Both types of environments have their specific application scenarios.

Currently, a lot of projects engage in the configuration and adaptation of pervasive applications. However, these systems focus on only one type of scenario, either on infrastructure-based Smart Environments [9],[10],[12],[13], or on infrastructure-less Peer-to-Peer scenarios [2],[6]. Thus, they normally do not provide efficient support for *both* system types. In this regard, our system *PCOM* [2] does not represent an exception, as it was initially created to provide middleware support for self-organizing infrastructures in networked mobile systems. PCOM indeed can be used in both types of scenarios. However, it regards each device as being equal and, thus, does not take special care of infrastructure devices whose additional resources could be used for con-

figuration. Though, this can improve calculation speed and decrease communication overhead considerably.

The main contribution of this paper is a framework for enabling the efficient use of both infrastructure-based and ad hoc environments. This allows to support both scenarios in an efficient way and represents an important step towards hybrid application configuration which is taken place on a subset of the involved devices. To provide support of infrastructure and ad hoc scenarios, we introduce a new concept called *Virtual Container*. This concept enables the local emulation of remote devices and allows local access to the relevant configuration logic on these devices. As the use of all elements of a programming language is privileged in complex scenarios and since we assume cooperative users behaviors, we decided to use mobile code for obtaining the distributed configuration logic. This enables centralized application configuration on distinguished devices without the necessity of remote validation of the obtained configuration, and it reduces communication overhead and configuration latency. In order to identify the device that performs centralized configuration and acts as a coordinator, we introduced a clustering scheme to our framework. Moreover, we provide several strategies to access the required distributed configuration logic for further reduction of the configuration latency, and we present an advanced mechanism which actually performs the process of calculating a valid application configuration. This mechanism uses the Virtual Container concept and the provided access strategies. As our evaluation results will show, this framework significantly reduces the configuration latency noticeable by the user.

The rest of this paper is structured as follows: In the next section, we discuss some related projects and point that these projects do not cover the full spectrum of possible environments. In Section 3, we state the prerequisites for this work. Then, we present our concepts and their realization in Sections 4 and 5. This is followed by our evaluation results. Finally, Section 7 concludes this paper and gives a short outlook on our future work.

## 2. RELATED WORK

Many projects deal with the development of abstractions which allow automatic adaptation of pervasive applications at runtime. In this section, some of them are presented.

Gaia [12] is a project which provides a CORBA-based middleware for Smart Environments, *Gaia OS*. It supports developers by providing services for the development of user-centric, resource-dependent, and context-aware mobile distributed applications. Gaia represents a highly integrated environment and supports various kinds of devices, such as audio devices, video cameras, or even fingerprint sensors. The system is transparent to the user, but yet regards security aspects. It supports stationary as well as mobile devices, but it is not suited for the use in ad hoc environments.

The project Aura [13] provides a highly integrated environment for pervasive computing that consists of various modules. Besides mobile devices like laptops, it seamlessly integrates existing infrastructure to support users in their daily work. In this connection, a context observer recognizes changes in the current user context and reports them to the task manager which exploits this information to adapt the system to the changed conditions. Such as Gaia, Aura cannot be used in Peer-to-Peer environments.

Another project that depends on resource-rich infrastructure devices is Matilda's Smart House [10] which focuses on Pervasive Healthcare support for aged persons. The system creates a lucent environment for users via mobile sensors and end-user devices. Therefore, it relies on the OSGi[1] platform for component and lifecycle management.

The iRoom project [9] aims at an increased user experience of distributed visualization, as it focuses on integrating high-resolution displays into application configuration. The main component of this project is the *iROS* middleware that provides services for the implementation of distributed applications by the use of infrastructure devices. iRoom supports the automatic adaptation of graphical user interfaces depending on the available devices, e.g., PDAs or laptops.

P2PComp [6] is a project to provide context-aware mobile devices by a Peer-to-Peer pervasive computing middleware. Therefore, the P2PComp middleware also uses the OSGi platform and extends it with an abstract communication layer. This project focuses on ad hoc environments and does not take special care of additional infrastructure devices, which yields suboptimal results in Smart Environments.

Our system PCOM [2] is a component system which was, similar to P2PComp, developed as a middleware for self-organizing infrastructures in mobile ad hoc networks without supporting infrastructure devices. However, PCOM provides automatic adaptation on system level, while P2PComp depends on additional user interaction. PCOM supports a wide range of end-user devices and can integrate additional infrastructure devices, but it does not take special care of their computation power and, hence, does not exploit available resources efficiently in many scenarios. Therefore, we developed a framework which we present in this paper.

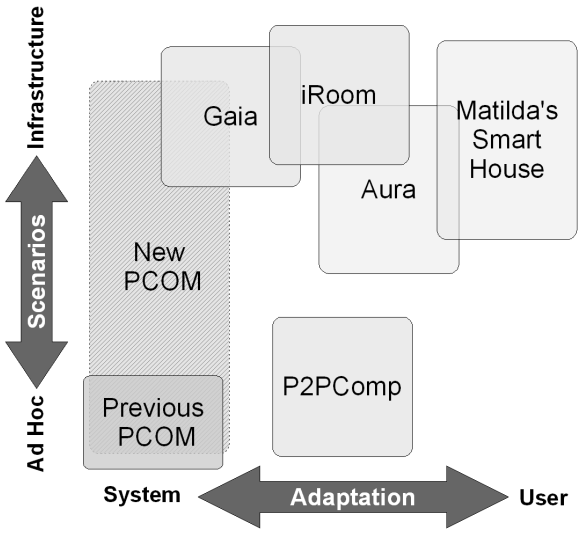


**Figure 1: Classification of PCOM & related projects**

In summary, one can see that some projects like Gaia or Aura focus on Smart Environments which means they rely on additional infrastructure, while systems like P2PComp or the PCOM system as yet concentrate on ad hoc scenarios. We created a survey of the discussed projects which is shown in Figure 1 and clarifies that only our new PCOM system provides system-level support for ad hoc as well as for infrastructure scenarios.

[1] **O**pen **S**ervices **G**ateway **i**nitiative

# 3. PREREQUISITES

In this section, we at first motivate our work. Then, we provide required background information about the PCOM system and the automatic configuration of distributed applications in PCOM. Finally, we briefly present an exemplary application scenario for PCOM.

## 3.1 Motivation

In infrastructure-less Peer-to-Peer environments, distributed application configuration is mandatory to guarantee applicability in each application scenario. All devices are involved in calculating a valid configuration in a distributed way which balances configuration load on them.

However, most realistic pervasive application scenarios are situated in heterogeneous environments where additional computation power of resource-rich stationary devices with external power supply is available. If such infrastructure devices perform centralized application configuration, they lessen configuration load for the mobile devices. Consequently, the operating time of these mobile devices increases. Furthermore, resource-rich devices can perform calculation of application configurations much faster due to their high computation power. This helps to decrease latencies and, thus, increases quality of service for the application user. In addition, the message complexity is linear to the number $n$ of devices if a configuration is locally calculated on a particular device, as all devices only have to communicate with this single device. However, completely distributed application configuration requires a message amount of $c \cdot \binom{n}{2} = c \cdot \frac{n \cdot (n-1)}{2}$ as each of the $n$ devices are connected with each other and every device needs to transmit a fixed number of $c$ messages. This yields a higher message complexity of $O(n^2)$. Thus, it can be seen that centralized configuration on resource-rich stationary devices implies many advantages.

In the following, we focus on the efficient support of these *Smart Environments* that feature resource-rich infrastructure devices as well as resource-weak mobile devices. In consequence, our approach has to efficiently support both infrastructure-based and infrastructure-less environments.

The concepts which we recognized to enable this support are presented in Section 4.

## 3.2 System architecture

PCOM is a component system for automatic adaptation and configuration of distributed pervasive applications [2]. It originally was designed for use in mobile ad hoc networks without additional infrastructure. Thus, PCOM is a system that does not rely on any central instance. This has serious impacts on PCOM's inital system architecture, which is illustrated in Figure 2.

PCOM provides a runtime environment for *components* with contractually specified interfaces for the design of distributed applications that are automatically configured and adapted during runtime without user interaction. Cooperative user behavior is assumed, i.e. all users are trustworthy.

The components are executed within a PCOM *Container* that provides a specified interface to the PCOM middleware and basic services for the components. A component features *factories* which are representatives for locally installed components and support a negotiation phase that recursively determines the non-functional parameters of a component without starting it. This enables a resource-conserving configuration of the components. In order to support contracts which represent resource dependencies between components, PCOM features *allocators* that encapsulate the access to exclusive resources (e.g., a single display) and, thus, allow transparent access to these resources. Allocators are used both during configuration and execution process of an application. Such as factories, they have to be registered on the corresponding container which then can determine a valid configuration for the distributed application, and execute the application.

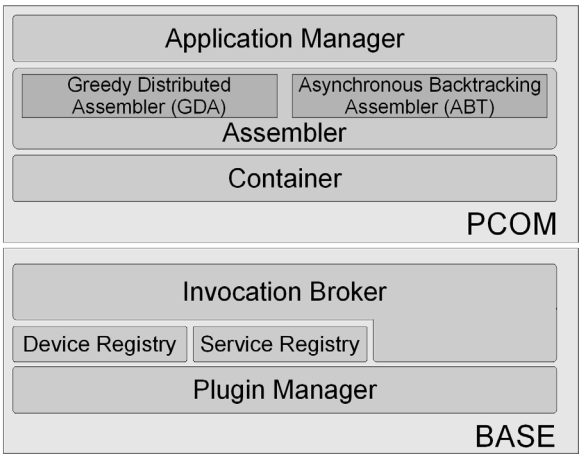


**Figure 2: Previous system architecture**

The automatic configuration of components is performed by so-called *assemblers* which enable access to components prior to their instantiation and, thus, decouple the configuration processes from the lifecycle management of the components. Currently, PCOM supplies a completely distributed assembler based on the Asynchronous Backtracking algorithm (*ABT*) by Yokoo et al. [15], and a distributed greedy heuristic (*Greedy Distributed Assembler, GDA*) [7]. These assemblers are optimized for use on resource-poor mobile devices. Lifecycle management of applications is performed by the *Application Manager* which allows to start, stop, and configure the distributed application with the aid of the containers and the assemblers.

PCOM uses the *BASE Microbroker* [3] to support automatic configuration and adaptation of communication protocols at runtime. This enables a more stable and flexible communication platform. BASE provides distribution-independent access to the offered services and decouples the application from the underlying communication protocols. The main component of BASE is the *Invocation Broker* which delegates method calls to the corresponding services on the mobile devices. Furthermore, BASE manages the devices in range through the *Device Registry* and the services available on a device through the *Service Registry*. The automatic configuration of the supported protocols is possible by the *Plugin Manager*, as the protocols are outsourced in plugins which are loaded and configured at runtime. The plugin concept of BASE allows the microbroker to support a wide range of end-user devices that reaches from simple microcontrollers and mobile phones up to full servers. The only requirement is the presence of a Java Virtual Machine that supports the CLDC[2] profile.

[2]**C**onnected **L**imited **D**evice **C**onfiguration, [14]

## 3.3 Automatic Application Configuration in PCOM

A PCOM application consists of one or more *components* that are independent from each other and atomic with respect to distribution. The application is modeled as a tree of interdependent components which are units of composition with contractually specified interfaces and explicit context dependencies. Components are recursively arranged within the tree. So, they may use other components to provide their service. PCOM components enclose directed *contracts* that describe on the one hand the functionality offered by the parent component, and on the other hand the requirements that need to be fulfilled by the child component.

Automatic configuration denotes the task of automatically determining a composition of components that can be instantiated simultaneously as an application. Such a composition is subject to two classes of constraints: structural constraints that define what constitutes a valid composition regarding functionalities, and resource constraints due to limited resources. The complexity of automatic configuration arises from the fact that both constraints must be fulfilled simultaneously. The components of an application form a tree structure of dependencies. To obtain a valid configuration, the components have to recursively resolve the contractually specified dependencies. As proved in [7], the problem of finding a possible configuration can be regarded as a *Distributed Constraint Satisfaction Problem (DCSP)* [15]. The result of a successful configuration process is a tree structure that comprises the involved resources and components.

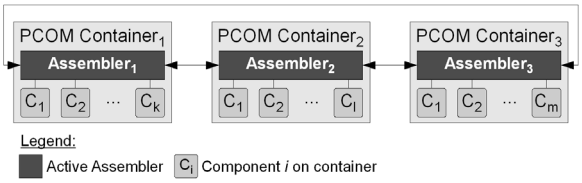


**Figure 3: Distributed configuration with PCOM**

Figure 3 presents distributed configuration with PCOM using GDA or ABT in a scenario with three containers. It can be seen that the assemblers of the involved containers perform configuration cooperatively. They use BASE for message exchanges. As mentioned above, completely distributed configuration causes a message complexity of $O(n^2)$. In the previous PCOM version, the user has to define statically which assembler shall be used for configuration.

## 3.4 Exemplary PCOM application

A pervasive presenter [8] represents a typical PCOM application scenario. The presenter leverages the resources present in the environment for a presentation application. The functionality of displaying presentation slides on remote systems is provided by the cooperation of distributed devices. The actual composition of available input and output devices is calculated by the PCOM configuration algorithms. Furthermore, this application composition can automatically be adapted by PCOM without user intervention.

# 4. CONCEPTS

In order to provide the possibility of adapting the distribution of automatic application configuration and efficiently support various environments, several changes are necessary to systems that was originally designed for completely distributed application configuration. In particular, we recognized four issues that have to be considered in the design of a framework to achieve the above mentioned functionality:

**1. Election of the device to perform centralized application configuration:** Before centralized configuration can take place, a dedicated device has to be identified which performs the actual configuration and acts as a coordinator for the configuration process. Therefore, the properties of the available devices have to be investigated to define this device. This has to be done in a distributed manner to retain applicability in Peer-to-Peer environments. The election of a specific node to become a coordinator for a group of nodes is typically done by the introduction of clustering schemes and is a common subject in the research area of mobile ad hoc networks. Consequently, a lot of schemes for organizing nodes to clusters exist [16]. Thus, we decided to develop a *Clustering Framework* which uses a common clustering algorithm of Basagni [1] to obtain the *cluster head* that acts as a coordinator for the *cluster members*. As clustering has to be performed before configuration, this process causes additional latencies. In our case, a valid configuration has to be calculated as fast as possible to ensure a high quality of service for the application user. Thus, the used clustering scheme should guarantee stable clusters and particularly avoid many reclustering processes. Therefore, the clustering strategies are of particular interest. We present the *Clustering Framework* we developed in Section 5.3.

**2. Recovery of configuration-specific information:** After the cluster head was elected, this node needs to collect the environmental information that is relevant for configuration processes, i.e. the available devices and the resources present on them. Furthermore, the validation of an application configuration has to be performed after the configuration was calculated. If this validation can be performed completely local by the central coordinator, communication amount and configuration latencies are reduced. As this configuration logic is very complex and calls for all elements of a programming language, we decided to use *Mobile Code* to transmit those code parts of a program to the cluster head which contain the corresponding logic. This enables the emulation of *Virtual Containers* at the cluster head, as Section 5.1 will show. Furthermore, the code to be loaded represents only small amounts of data which do not make a noteworthy contribution to the configuration latencies, as the evaluation section will show. Further details concerning the collection of data relevant for configuration is given in Sections 5.5 and 5.6.

As device resources are typically limited, changes in the availability of these resources have to be transmitted to the cluster head which then updates the local proxy of the remote device. To notify the cluster head of changed resource conditions, we implemented a distributed *Event Service* which is presented in Section 5.2.

**3. Automatic adaptation of the degree of decentralization:** The optimal distribution of the configuration process among the available devices depends on the resources that are available on these devices. Thus, a mechanism that automatically determines which devices should be involved in the process of calculating a valid configuration is required. We present an abstraction that enables this adaptation in Section 5.4. This mechanism at first identifies the current

resource condition in the environment in a distributed manner. Subsequently, it decides if the configuration has to be performed in a centralized or in a distributed fashion, and which device(s) are involved in the process for obtaining a valid configuration. Further extensions to this abstraction are possible to support additional configuration strategies, e.g., hybrid configuration on arbitrary subsets of all devices.

**4. Access of configuration-specific information in advance to decrease latencies:** Another issue of relevance is the point in time at which the remote configuration logic is accessed and transmitted to the cluster head. Therefore, it is necessary to have a look on the *interaction model* of a pervasive application which is illustrated in Figure 4a.

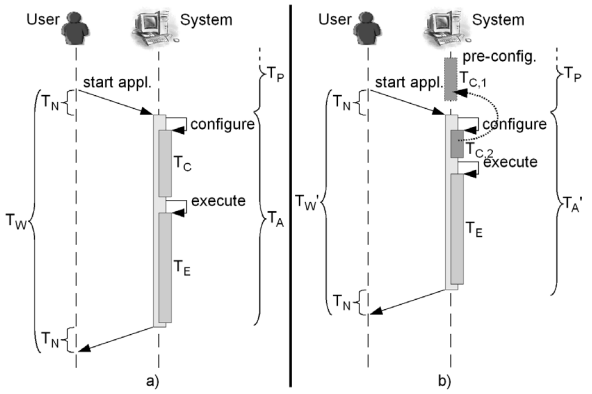


**Figure 4: a) Interaction diagram of previous PCOM, b) Interaction diagram with use of pre-configuration**

In this model, the *total waiting period* $T_W$ for the user is composed of the *application time* $T_A$ and the *network latency* $T_N$ which arises twice. The application time can further be divided into the configuration time $T_C$, and the execution time $T_E$. Thus, $T_W$ can be expressed by the following Equation 1.

$$T_W = 2 \cdot T_N + T_C + T_E \quad (1)$$

Hence, $T_W$ can be reduced if, for instance, another type of network technology is used, or if the configuration or execution of the application is speeded up. Previous work [7] aimed in minimizing $T_C$ and $T_E$ and represents a good basis for further improvements. However, optimizations have only been taken within the application time $T_A$. But in practice, the period $T_P$ before the execution of the application could be used to decrease configuration latency. In this period, the proactive transmission of configuration-relevant data could be performed without increasing $T_W$ for the user. Thus, the waiting period for the user can be reduced by reorganizing some part of the configuration process into the period $T_P$, as it is illustrated in Figure 4b. This leads to a reduced configuration time of

$$T_{C,2} = T_C - T_{C,1}$$

within $T_A$. Both $T_A$ and $T_W$ are reduced to $T'_A$ and $T'_W$.

Thus, we focused on an access method to obtain the configuration logic in advance in order to reduce the configuration latency. Furthermore, we designed an access method that supports possible reclustering processes and helps to decrease latencies in these cases. Section 5.5 presents the realized access methods.

# 5. REALIZATION

Figure 5 gives an overview of PCOM's and BASE's new system architecture. We present the changes to the architecture in the following sections.

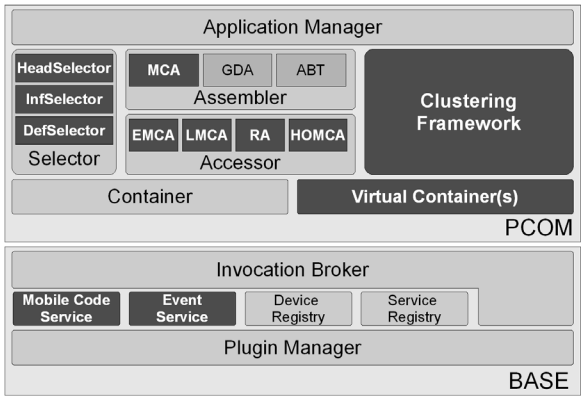


**Figure 5: New system architecture. The new elements of the architecture are highlighted in black.**

## 5.1 Virtual Containers

The main goal of the work underlying this paper was the efficient use of the particular computation power on resource-rich infrastructure devices that can serve as a cluster head in Smart Environments. The cluster head performs configuration and represents the coordinator for all adjacent devices which, thus, constitute the cluster members. This means that the resources used for configuration are hosted on the cluster head, while the logic for validating a configuration is distributed among the devices involved in a configuration process. This presumes that the cluster head acquires knowledge of the currently available components and resources on the other devices. Thus, as already mentioned in our motivation, we decided to transmit the configuration logic to the cluster head in order to perform centralized configuration for the cluster members. As resources are limited and PCOM enables access to these resources via allocators, changes in the state of remote allocators and factories have to be considered. Furthermore, both allocators and factories need certain standard services from the PCOM containers they are registered on. Thus, we introduce the *Virtual Container (VC)* concept, illustrated in Figure 6. This concept is based on the idea that the cluster head emulates a "virtual" PCOM Container for every cluster member which simply contains the functions relevant for configuration, and keeps the device states of the remote devices up to date. This enables a strong decoupling of the configuration processes from the real PCOM Containers. Based on this concept, methods for eager and lazy loading of the configuration logic are presented in Section 5.5.

To obtain the required containers as well as the factories and allocators that are registered at these containers, we decided to use mobile code [4]. It has to be considered that mobile code normally is subject to security risks, as the transmission of threats such as viruses or worms is potentially possible. However, regarding the system model we presented in Section 3.2, we assume cooperative user behaviors, which enables the use of mobile code without the necessity of using digital signatures and, thus, the manage-

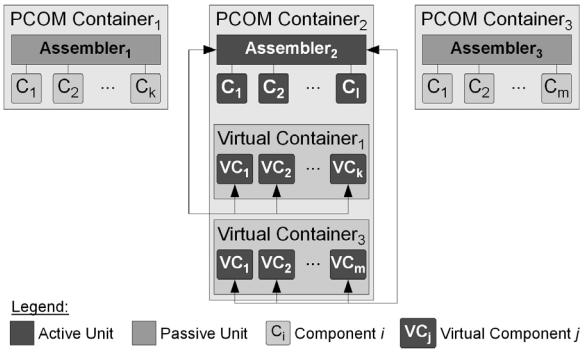


**Figure 6: Virtual Container concept**

ment of certificates. Mobile code enables the transmission of code segments from one runtime environment to another. The only requirement is the existence of a class loader [11] which is the case if a device supports CLDC specification.

There exist multiple paradigms to obtain remote code. [4] presents three paradigms besides the classical client-server model where code is only executed on a server: *Remote Evaluation*, *Code On Demand*, and *Mobile Agents*. In our case, Code On Demand is the preferrable choice, as it represents the scenario that a process $A$ on device $S_A$ possesses the required resources for configuration and loads the executable code of process $B$ on device $S_B$ to execute it locally. Furthermore, this code may be effectively constrained to solely executing uncritical actions if Code On Demand is used.

## 5.2 Extensions to BASE

As one can see in Figure 5, there are two architectural extensions to BASE:

- **Event Service:** The basic idea of the VC concept presented in Section 5.1 is the creation of one virtual container for every remote PCOM Container. This VC emulates the remote container. As availability of resources on remote containers may change over time, the VCs have to be updated. This happens via BASE's new Event Service which implements a distributed message service.

- **Mobile Code Service:** In order to update the state of a remote container, the configuration logic has to be transfered to the current cluster head. This is possible using the *Mobile Code Service* of BASE. This service is implemented in a way that it can transfer general classes which also enables the distribution of BASE plugins, for instance.

## 5.3 Clustering Framework

The *Clustering Framework* is a main component of the new PCOM system architecture. The central component of this framework is the *Cluster Service* which provides methods for requesting the current cluster head and the cluster members. It delegates occurring events (e.g., neighborhood changes) to the corresponding *clustering strategies*.

Regarding these clustering strategies, we have implemented the *Distributed Mobility Adaptive Clustering (DMAC)* algorithm that represents a completely distributed version of the *Generalized Clustering Algorithm* of Basagni [1]. In this algorithm, adjacent nodes initially exchange their node weights. The node with highest weight becomes cluster head. According to custom, weights are assigned values between 0 and 1. As we described previously in this paper, many cluster changes should be avoided as they cause additional latencies due to the reclustering process. This means the clustering strategies should be based on node properties which do not or only marginally change over time. Thus, we developed several clustering strategies which calculate the node weight based on the devices' hardware that is relevant for computation and the power supply of the devices. This typically leads to resource-rich cluster heads and decreases the probability of inevitable reclusterings because the cluster head ran out of power. As computation resources typically do not change for the short term, and infrastructure devices as likely cluster heads are connected to external power sources, this results in high cluster stability particularly in Smart Environments. The implemented clustering strategies are the following:

- The battery-awareness strategy (**BAStrategy**) determines based on the remaining power sources of a node which weight $w_{BA}$ it is assigned. We decided to calculate the weight based on two factors: The remaining power of the device's internal battery power, and if the device is connected to an external power source. Thus, the node weight calculates to

$$w_{BA} = 0.5 \cdot p_{bat} + 0.5 \cdot AC\_connected,$$

  where $p_{bat}$ represents the remaining battery power relative to full charge, and $AC\_connected$ is a binary variable with $AC\_connected = 1$ in case the device is connected to an external AC power source, otherwise $AC\_connected$ becomes zero. In case a node solely depends on an external power source which is likely for stationary devices, $p_{bat}$ is assigned one as the external power source is supposed to be continuously available. Hence, a node that is connected to an external power source gets the highest possible weight of 1.0, while mobile devices without external power source are assigned weights of 0.5 or less, as they are more likely to breakdown soon.

- The resource-awareness strategy (**RAStrategy**) assigns weights $w_{RA}$ based on the available computational resources, i.e., the CPU. Therefore, nodes are assigned specific power classes, based on the CPU frequency and the number of CPU cores. We decided to choose a simple strategy that assigns the maximum weight of 1.0 to a powerful dual core CPU with a clock frequency of 2.0 GHz or more, while embedded devices with single core CPUs of less than 500 MHz clock frequency are assigned a weight of 0.0. Accordingly, devices with CPUs between these bounds get graded weights.

- The implemented strategy that is based on device types (**DTStrategy**) assigns weights $w_{DT}$ according to the type of the device. Therefore, we defined the device types "infrastructure" for devices like desktop PCs, notebooks, or servers ($w_{DT} = 1.0$), "mobile" for smart phones and PDAs ($w_{DT} = 0.5$), and "cellular" for simple cellular phones ($w_{DT} = 0.0$). Of course, additional device types and weightings are possible.

- Finally, the combined strategy (**CombStrategy**) is an advanced strategy which uses combined weighted sums of the weights from the previously described strategies. This proceeding is similar to that presented in [5] and leads to the following Equation 2 for the combined weight $w_{Comb}$.

$$w_{Comb} = \frac{1}{\sum_{i=1}^{3} x_i} \cdot (x_1 \cdot w_{BA} + x_2 \cdot w_{RA} + x_3 \cdot w_{DT}) \quad (2)$$

  In this equation, $x_1$ to $x_3$ are balancing parameters with $0 \leq x_i \leq 1$ which allow a different weighting of the different strategies. If only some of the above strategies should be considered, the corresponding parameters $x_i$ of the other strategies have to be set to zero. Naturally, this strategy can be extended with additional strategies which simply have to be added as weighted additional summands to Equation 2.

The *Cluster Context* provides methods for the transmission of messages to adjacent nodes and manages the group membership of adjacent nodes. Furthermore, the *Cluster Context* stores strategy-dependent node properties and the current internal state of the group formation. The possible states are *cluster creation* (while the cluster head is being elected), and *cluster maintenance*. In the maintenance state, the cluster head updates the cluster state if a new device appears or a previously available device disappears. Reclustering is only necessary if the current cluster head is no longer present, or in case of a newly available device that is assigned the highest weight.

## 5.4 Selector abstraction

As mentioned in the motivation for this work, adapting the distribution of automatic application configuration requires a mechanism to allow the automatic selection of an assembler suited for application configuration in a specific environment. Therefore, we introduce the *Selector* abstraction that enables the automatic selection of arbitrary assemblers on any container. We introduced three different strategies:

- **DefaultSelector:** This strategy uses a preassigned assembler and starts it. Its behavior is equivalent to the previous PCOM implementation and is particularly suited for use in Peer-to-Peer Environments.

- **HeadSelector:** In this variant, a preassigned assembler is always started on the current cluster head. This selector is particularly useful in Smart Environments.

- **InfSelector:** Using this strategy, the selector at first verifies if the chosen cluster head is a stationary infrastructure device. In this case, the MCA is executed on the cluster head. Otherwise, GDA is used for application configuration. Thus, this selector is suited for both types of environments.

Of course, the implementation of additional strategies, e.g., for hybrid application configuration, is possible. This represents a flexible solution for supporting various homogeneous as well as heterogeneous pervasive computing scenarios.

## 5.5 Mobile Code Accessor Framework

In order to allow the loading of remote classes via mobile code, we realized a framework to access the required classes. The main part of this *Mobile Code Accessor Framework* are various so-called *Accessors* to allow different access methods.

Those parts of the original *Allocators* and *Factories* which are relevant for configuration processes were outsourced into new serializable configuration classes. This enables their transmission to the cluster head which then can locally calculate a valid configuration. As resources typically are limited, changes in the state of remote resources have to be transmitted to the cluster head. This happens by the previously described BASE *Event Service* that represents a distributed event listener. Every configurator has to report changes in its state by events. To enable access to context objects on remote containers for the cluster head, we introduced the *Virtual Container* concept described in Section 5.1, where every configuration class is administrated within its own virtual container.

Following, we focus on the realized access methods. We implemented four different *accessors* which represent several strategies for obtaining the required configuration logic.

- **Remote Accessor:** This accessor simply delegates requests by Remote Invocation to remote containers and does not use mobile code. Thus, an assembler that uses this access method behaves exactly like the previous PCOM assemblers.

- **Eager Mobile Code Accessor:** This accessor loads configuration classes from cluster members in advance, as soon as they are in communication range. Thus, it implements the pre-configuration process presented in Section 4. In case a cluster head changes, the old head unloads all classes to deallocate memory, as they are no longer needed. This yields the advantage that the system performs proactive class loading to reduce the latency in a possibly following application configuration. One drawback arises for highly dynamic scenarios, or if the cluster head frequently changes which leads to increased communication overhead. This drawback can be reduced by choosing a suitable cluster strategy such as the DTStrategy (compare Section 5.3) or the use of the *Handover Mobile Code Accessor* (see below).

- **Lazy Mobile Code Accessor:** In this method, the required classes are not loaded until the time they are needed. This complements with eager class loading and implicates that the cluster head's resources are conserved. Furthermore, the traffic load of the network is reduced. A disadvantage of this method is the increased configuration latency for the user, as the class loading happens during the user interaction period.

- **Handover Mobile Code Accessor:** Frequent cluster head changes can significantly burden the network in case of eager class loading. To reduce this network load, we have implemented an additional *Handover Mobile Code Accessor* which performs eager class loading. In case the cluster head changes, this accessor immediately transmits the previous cluster head's state to the new cluster head. Consequently, the new cluster head does not need to inquire the configuration classes from the involved devices. This helps to decrease configuration latency, as the evaluation section will show.

**Table 1: Survey of the devices used for evaluation**

| Device Type | CPU | RAM | HD | WLAN | OS | Java VM |
|---|---|---|---|---|---|---|
| Notebook | Pentium M 1.6 GHz | 1.0 GB | 120 GB | 802.11g | WinXP SP 2 | Sun J2SE with JVM v1.6 |
| Smart phone | PXA 270 (520 MHz) | 48 MB | 44 MB | 802.11b | WinCE 5.1 | IBM J9 v2.2, Found. 10 Profile |
| PDA | PXA 255 (400 MHz) | 128 MB | 17 MB | 802.11b | WinCE 4.2 | IBM J9 v2.2, Found. 10 Profile |

## 5.6 Mobile Code Assembler

The *Mobile Code Assembler (MCA)* represents a part of the presented *Virtual Container* concept. It is an assembler that uses mobile code to get the needed configuration logic.

A component is configured if it finds all required resources or additional components. The participating configuration classes contain a `configure()` method that is recursively called for every element of the tree. Each class of the application tree configures itself and then calls the `configure()` method of its child components. Finally, the obtained configuration is serialized into a particular object which then can be transmitted to other assemblers.

# 6. EVALUATION

The evaluation section particularly deals with the question if the virtual container concept really enables the reduction of configuration latencies that are noticeable for the user, and in which scenarios this is the case. After presenting the evaluation setup, we show the results of a Smart Environment and an ad hoc scenario.

## 6.1 Experimental Setup

To obtain the following results, we used a laptop, four smart phones, and a PDA with the specifications as denoted in Table 1. The Java VMs on the smart phones and the PDA support the class loading feature, which enables the use of the Mobile Code Accessors on these devices.

The exemplary PCOM application represents a binary tree of depth 3, i.e. the application consists of $2^4 - 1 = 15$ components. Therefore, at least 15 resources have to be present on the available $n$ devices. Thus, we put an amount of $R(n) = \lceil \frac{15}{n} \rceil$ resources on every device to guarantee that a valid configuration exists. According to the motivation of the work, we evaluated the new *Mobile Code Assembler* in comparison to the previously used *Greedy Distributed Assembler* both in an infrastructure and an ad hoc environment to show the efficient support of both types of scenarios.

The ad hoc scenario consisted of the PDA and up to four smart phones. One of smart phones was elected cluster head, as they have a CPU with a higher clock frequency than the PDA. In the infrastructure scenario, we used the laptop instead of one of the smart phones to represent a resource-rich device. Hence, the notebook became cluster head there. In both scenarios, we used the 802.11 WLAN Ad Hoc mode to connect the involved devices with each other. The test application was started on the PDA, respectively, as this represents the resource-weakest device.

In the Smart Environment, the laptop became cluster head. We did not consider latencies due to reclustering processes there as they are very unlikely due to the continuous presence of the stationary devices and the implemented clustering strategies that guarantee almost fixed weights for stationary devices. Thus, reclustering latencies would falsify the evaluation results in Smart Environments.

**Table 2: Message sizes**

| Message / Class | Type | Size [byte] |
|---|---|---|
| *RequestInfo* | Clustering (DMAC) | 335 |
| *Join* | Clustering (DMAC) | 285 |
| *AcquireHead* | Clustering (DMAC) | 292 |
| *ComponentConfig* | Configuration Class | 8.767 |
| *CapabilityConfig* | Configuration Class | 5.123 |
| *Subscribe* | EventService Mess. | 399 |
| *Unsubscribe* | EventService Mess. | 401 |
| *Event* | EventService Mess. | 335 |

## 6.2 Evaluation results

In the following, we present our evaluation results, beginning with space and communication overheads. Then, we focus on the emerging latencies for cluster formation and class loading. This is followed by configuration latency measurements in an infrastructure-based *Smart Environment* and in an infrastructure-less *Peer-to-Peer Environment*. To obtain the following evaluation results, we took 20 measurements at each scenario. The values shown in the figures represent the mean values of these measurements, respectively.

### 6.2.1 Message and code overhead

A significant requirement to the framework was the gentle use of the limited system resources on mobile devices. The total space overhead of our framework on the devices is 131 kB. This overhead is acceptable, compared to the available space on the used mobile devices presented in Table 1.

Regarding communication overhead, the message sizes of the distinct clustering and event service messages and the configuration classes are specified in Table 2. Usually, the cluster head loads the configuration classes of its $(n-1)$ cluster members and registers at the *Event Service* of the cluster members for two events that notify about changed total and free resources on this device, respectively. As we used 802.11b radio technology, the maximum effective data rate is about 50 % of 11 Mbit/s. Since the required configuration classes have a size of around 14 kB, the actual process of transmitting this amount of data requires only few milliseconds and represents a small fraction compared to the hardware and software latencies arising during the class loading process, as it will be seen. This retains applicability of the Mobile Code concept. Furthermore, regarding the message and class sizes, the transmitting medium has enough capacity for the required communication between cluster head and cluster members. Correspondingly, the cluster head's CPU is the restrictive factor for application configuration.

### 6.2.2 Initialization

As the realized configuration process uses the presented *Virtual Container* concept, the required classes have to be loaded via mobile code which causes additional latencies. This has to happen initially, and also after environmental

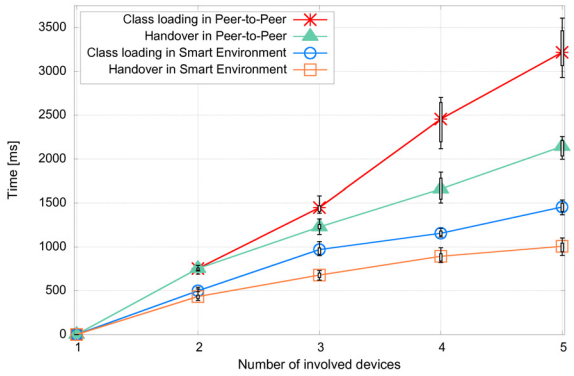


**Figure 7: Class loading latencies**

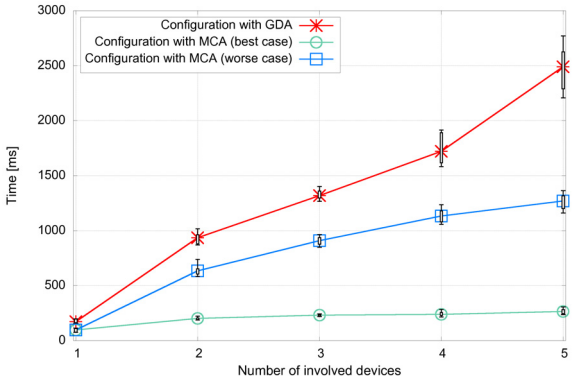


**Figure 8: Latencies in Smart Environment**

changes, e.g. the presence of new devices, have taken place. These latencies are presented in Figure 7.

The figure states that class loading in the Smart Environment compared to the ad hoc environment is performed faster by 33 % in case of two devices, and up to 55 % if five devices are involved. This figure also displays that the use of a handover mechanism like the implemented *Handover Mobile Code Accessor* reduces class loading latencies up to 33 % in an ad hoc environment with five involved devices, and around 31 % in the corresponding infrastructure scenario. Thus, the use of a handover mechanism can significantly reduce class loading times, particularly when the number of involved devices rises.

### 6.2.3 *Configuration in Smart Environment*

Figure 8 shows the configuration latencies in Smart Environments. We compared the new *Mobile Code Assembler* (MCA) with the *Greedy Distributed Assembler* (GDA). Regarding MCA, we distinguished between two cases:

- If the required configuration classes have already been loaded in advance, only configuration of the application is necessary. This represents the best case for MCA configuration.
- In case of lazy class loading or if an environmental change has just taken place when a configuration process is initiated, MCA additionally needs to obtain the corresponding configuration logic. Thus, the configuration process comprehends class loading *and* the configuration itself. This is the worst case for the MCA.

It can be seen that in every single measurement, the MCA calculated a valid configuration much faster than the GDA. Regarding the best case measurements for MCA, configuration latency added up to only a small fraction of GDA latency, as it is almost independent from the number of involved devices. MCA outperforms GDA by up to 89 % in case of five devices. This is founded in the fact that MCA configuration can be performed completely local by the cluster head, as the required configuration logic was obtained before and, hence, the cluster head's CPU can be used efficiently as it does not have to wait for I/O operations. If the *Eager Mobile Code Accessor* is used, this significant advantage is really achievable, as the cluster head can immediately calculate and validate a suitable configuration completely locally as the required classes were loaded in advance. Furthermore, as the configuration logic only has to be updated in case of environmental changes, MCA's advantage is particularly large in scenarios with a low degree of dynamics. Comparing GDA configuration with MCA configuration's worst case measurements, the latency of the MCA was still lower by 10 % up to 22 %. This means that even in highly dynamic scenarios with fluctuating availability of devices which necessitates class loading before configuration, MCA outperforms GDA. Hence, MCA should be prefered over GDA configuration whenever a resource-rich device is available as this leads to considerable performance gains.

### 6.2.4 *Configuration in Peer-to-Peer Environment*

Besides the Smart Environment measurements, we evaluated our framework in an ad hoc scenario where we also compared GDA configuration with both best case and worst case MCA configuration. Here, the worst case latencies also comprise the reclustering process, as it is more likely that the cluster head changes in a Peer-to-Peer Environment due to the higher degree of dynamics in this scenario. The results concerning configuration latencies are displayed in Figure 9. The figure states that best case MCA configuration is between 52 % and 61 % faster than GDA configuration. If the classes still need to be loaded before the configuration, MCA performs worse than GDA by 10 % up to 23 % regarding configuration latencies, and even by 19 % up to 33 % in the worst case, when an additional reclustering is necessary. The reduced advantage of MCA in ad hoc environments can be explained by the fact that the cluster head was much resource-weaker than in the Smart Environment measurements. Consequently, the calculation of a valid configuration needed more time in this scenario.

Recapitulating, MCA configuration in Peer-to-Peer environments only makes sense if class loadings are scarce as the sum of class loading and configuration process takes more time than GDA configuration. Reclustering even increases latencies. In case of dynamic scenarios which involve many class loading processes, configuration with GDA should be prefered for obtaining minimum latencies. Thus, we suggest using the *InfSelector* strategy (compare Section 5.4) which causes configuration with MCA in infrastructure scenarios and configuration with GDA in Peer-to-Peer environments.

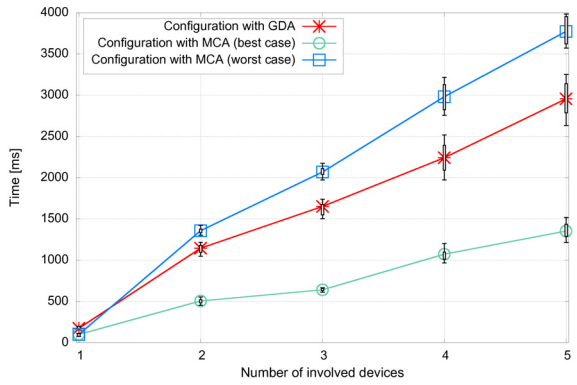


**Figure 9: Latencies in Peer-to-Peer Environment**

## 7. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a new approach that enables the efficient support of automatic application configuration both in infrastructure-based and in ad hoc pervasive environments. Main component of this approach is a *Clustering Framework* that enables cluster formation and the election of a cluster head according to typical criteria like resource-awareness or energy-awareness of the devices. We developed the new *Virtual Container* concept that supports the use of *Mobile Code* to load configuration-specific classes to the cluster head and, hence, enables efficient centralized application configuration on resource-rich devices. Class loading in advance and handovers between changing cluster heads for further decrease of the configuration latency are provided by *Accessors*. The actual configuration is performed by the new *Mobile Code Assembler* which integrates the obtained Virtual Containers and avoids any communication latencies during configuration process. In the evaluation section, we proved that our framework can reduce configuration latencies significantly, particularly in Smart Environments and when eager class loading is used. This enables the desired efficient support of various environments.

Our next step is the design of an advanced *Hybrid Assembler* which enables configuration on arbitrary subsets of the available devices and, thus, in various environments. Main goal of this assembler is the optimized use of the available computation power spread on the present devices to minimize configuration delays. The framework we presented in this paper is an important step towards efficient hybrid application configuration in pervasive computing scenarios and will be used by this *Hybrid Assembler*.

## 8. ACKNOWLEDGEMENT

This work is funded by the German Research Foundation within DFG Priority Programme 1140 - Middleware for Self-organizing Infrastructures in Networked Mobile Systems.

## 9. REFERENCES

[1] S. Basagni. Distributed clustering for ad hoc networks. In *ISPAN '99: Proceedings of the 1999 International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN '99)*, page 310, Washington, DC, USA, 1999. IEEE Computer Society.

[2] C. Becker, M. Handte, G. Schiele, and K. Rothermel. PCOM - a component system for pervasive computing. In *Proceedings of the 2$^{nd}$ IEEE Conference on Pervasive Computing and Communications (PerCom 2004)*, pages 67–76, 2004.

[3] C. Becker, G. Schiele, H. Gubbels, and K. Rothermel. BASE - A Micro-Broker-Based Middleware for Pervasive Computing. In *Proceedings of the 1$^{st}$ IEEE International Conference on Pervasive Computing and Communications (PerCom 2003)*, Washington, DC, USA, 2003. IEEE Computer Society.

[4] A. Carzaniga, G. Picco, and G. Vigna. Designing distributed applications with mobile code paradigms. In *Proceedings of the 19$^{th}$ International Conference on Software Engineering*, pages 22–32, 1997.

[5] M. Chatterjee, S. Das, and D. Turgut. WCA: A weighted clustering algorithm for mobile ad hoc networks. *Journal of Cluster Computing, Special Issue on Mobile Ad hoc Networks*, 5, April 2002.

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

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

[8] M. Handte, S. Urbanski, C. Becker, P. Reinhard, M. Engel, and M. Smith. 3PC/MarNET Pervasive Presenter. *Demonstration at the 4$^{th}$ Int.'l Conference on Pervasive Computing and Communications (PerCom 2006)*, Pisa, Italy, July 2006.

[9] B. Johanson, A. Fox, and T. Winograd. The interactive workspaces project: Experiences with ubiquitous computing rooms. *IEEE Pervasive Computing*, 1(2):67–74, April–June 2002.

[10] C. Lee, D. Nordstedt, and S. Helal. Enabling smart spaces with OSGi. *IEEE Pervasive Computing*, 2(3):89–94, July–Sept. 2003.

[11] S. Liang and G. Bracha. Dynamic class loading in the Java virtual machine. In *Conference on Object-oriented programming, systems, languages, and applications (OOPSLA'98)*, pages 36–44, 1998.

[12] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R. Campbell, and K. Nahrstedt. A middleware infrastructure for active spaces. *IEEE Pervasive Computing*, 1(4):74–83, Oct.–Dec. 2002.

[13] J. P. Sousa and D. Garlan. Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environments. In *Proceedings of the 3$^{rd}$ Working IEEE/IFIP Conference on Software Architecture*, pages 29–43, August 2002.

[14] Sun Microsystems. Connected Limited Device Configuration (CLDC) Specification, Vers. 1.1, 2003.

[15] M. Yokoo, E. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):673–685, Sept.–Oct. 1998.

[16] J. Y. Yu and P. H. J. Chong. A survey on clustering in schemes for mobile ad hoc networks. In *Communications Surveys and Tutorials, IEEE*, volume 7, pages 32–48. IEEE, 2005.